# Some results about the variations of Higher-Order Logic underlying proof assistants

Andrei Popescu
Middlesex University London

Joint work with Ondřej Kunčar

# Part I. Motivation

# Proof Assistants

Software program for interactively

- modeling mathematical concepts
- and stating and proving theorems about them

# Proof Assistants

Software program for interactively

- modeling mathematical concepts
- and stating and proving theorems about them

They are the guardians of the galaxy!

# Proof Assistants

Software program for interactively

- modeling mathematical concepts
- and stating and proving theorems about them

They are the guardians of the galaxy!
But who's guarding the guardians?

# Proof Assistants

Software program for interactively

- modeling mathematical concepts
- and stating and proving theorems about them

They are the guardians of the galaxy!
But who's guarding the guardians?

Structure

- Logical core
- Tool layer
- User interface

# Proof Assistants

Software program for interactively

- modeling mathematical concepts
- and stating and proving theorems about them

They are the guardians of the galaxy!
But who's guarding the guardians?

Structure

- Logical core
- Tool layer
- User interface Bugs?

# Proof Assistants

Software program for interactively

- modeling mathematical concepts
- and stating and proving theorems about them

They are the guardians of the galaxy!
But who's guarding the guardians?

Structure

- Logical core
- Tool layer
- User interface  Bugs? OK

# Proof Assistants

Software program for interactively

- modeling mathematical concepts
- and stating and proving theorems about them

They are the guardians of the galaxy!
But who's guarding the guardians?

Structure

- Logical core
- Tool layer Bugs?
- User interface Bugs? OK

# Proof Assistants

Software program for interactively

- modeling mathematical concepts
- and stating and proving theorems about them

They are the guardians of the galaxy!
But who's guarding the guardians?

Structure

- Logical core
- Tool layer Bugs? OK
- User interface Bugs? OK

# Proof Assistants

Software program for interactively

- modeling mathematical concepts
- and stating and proving theorems about them

They are the guardians of the galaxy!
But who's guarding the guardians?

Structure

- Logical core Bugs?
- Tool layer Bugs? OK
- User interface Bugs? OK

# Proof Assistants

Software program for interactively

- modeling mathematical concepts
- and stating and proving theorems about them

They are the guardians of the galaxy!
But who's guarding the guardians?

Structure

- Logical core Bugs? Well, OK
- Tool layer Bugs? OK
- User interface Bugs? OK

# Proof Assistants

Software program for interactively

- modeling mathematical concepts
- and stating and proving theorems about them

They are the guardians of the galaxy!
But who's guarding the guardians?

Structure

- Logical core Bugs? Well, OK Logical flaws?
- Tool layer Bugs? OK
- User interface Bugs? OK

# Proof Assistants

Software program for interactively

- modeling mathematical concepts
- and stating and proving theorems about them

They are the guardians of the galaxy!
But who's guarding the guardians?

Structure

- Logical core Bugs? Well, OK Logical flaws? Not OK!
- Tool layer Bugs? OK
- User interface Bugs? OK

# Proof Assistants

Example implementation bug: Side-condition of inference rule not implemented correctly, i.e., as prescribed by the logical system

Example logical flaw: The termination/guardedness checker is conceptually flawed

Logical flaws should be very unlikely to occur in mature, heavily used proof assistants...

# Proof Assistants: Who's Guarding the Guardians?

Coq 8.4pl2 Maxime Dénès and Daniel Schepler (2013)
Hypothesis Heq : (False → False) = True.
Fixpoint contradiction (u : True) : False := contradiction ( match Heq in (_ = T)
return T with | eq_refl => fun f:False ⇒ match f with end end ).
Lemma foo : provable_prop_extensionality → False.
< four-line proof >

Isabelle 2013-2 Ondřej Kunčar (2014)
consts c :: bool
typedef T = {True, c} by blast
defs c_bool_def: c::bool ≡ if (∀(x::T) y. x = y) then False else True
lemma A: (∀(x::T) y. x = y) ⟷ c"
< one-line proof >
theorem False

< one-line proof >

Apologies to Agda, Dafny, PVS, etc. for the omissions...

# Proof Assistants: Who's Guarding the Guardians?

Well maintained, heavily used proof assistants
not immune to proofs of False

Coq 8.4pl2 Maxime Dénès and Daniel Schepler (2013)
Hypothesis Heq : (False → False) = True.
Fixpoint contradiction (u : True) : False := contradiction ( match Heq in (_ = T)
return T with | eq_refl => fun f:False ⇒ match f with end end ).
Lemma foo : provable_prop_extensionality → False.
< four-line proof >

Isabelle 2013-2 Ondřej Kunčar (2014)
consts c :: bool
typedef T = {True, c} by blast
defs c_bool_def: c::bool ≡ if (∀(x::T) y. x = y) then False else True
lemma A: (∀(x::T) y. x = y) ⟷ c"
< one-line proof >
theorem False

< one-line proof >

Apologies to Agda, Dafny, PVS, etc. for the omissions...

# Proof Assistants: Who's Guarding the Guardians?

Well maintained, heavily used proof assistants
not immune to proofs of False

Does this endanger the whole galaxy?

# Proof Assistants: Who's Guarding the Guardians?

Well maintained, heavily used proof assistants
not immune to proofs of False

Does this endanger the whole galaxy? No.

# Proof Assistants: Who's Guarding the Guardians?

Well maintained, heavily used proof assistants
not immune to proofs of False

Does this endanger the whole galaxy? No.

1. So far, such flaws have been successfully fixed.

# Proof Assistants: Who's Guarding the Guardians?

Well maintained, heavily used proof assistants
not immune to proofs of False

Does this endanger the whole galaxy? No.

1. So far, such flaws have been successfully fixed.
2. Honest, responsible users stay away from proofs of False.

# Proof Assistants: Who's Guarding the Guardians?

Well maintained, heavily used proof assistants
not immune to proofs of False

Does this endanger the whole galaxy? No.

1. So far, such flaws have been successfully fixed.
2. Honest, responsible users stay away from proofs of False.

-1. Automatic tools don't know what honesty means.

# Proof Assistants: Who's Guarding the Guardians?

Well maintained, heavily used proof assistants
not immune to proofs of False

Does this endanger the whole galaxy? No.

1. So far, such flaws have been successfully fixed.
2. Honest, responsible users stay away from proofs of False.

-1. Automatic tools don't know what honesty means.

-2. Certification authorities become reluctant.

Burkhart Wolff

I am [within] the EUROMILS project part of the team that attempts to get a common criteria (CC EAL5) evaluation for PikeOS through, where the models and proofs were done with Isabelle.
I had a lengthy debate with evaluators and (indirectly) BSI representatives which became aware [of a proof of False].

# Proof Assistants: Who's Guarding the Guardians?

Well maintained, heavily used proof assistants
not immune to proofs of False

# Proof Assistants: Who's Guarding the Guardians?

Well maintained, heavily used proof assistants
not immune to proofs of False

Most vulnerable place: the definitional mechanism

# Proof Assistants: Who's Guarding the Guardians?

Well maintained, heavily used proof assistants
not immune to proofs of False

Most vulnerable place: the definitional mechanism

Why is that?

# Proof Assistants: Who's Guarding the Guardians?

Well maintained, heavily used proof assistants
not immune to proofs of False

Most vulnerable place: the definitional mechanism Why is that?

Two approaches to theorem proving

- Axiomatic approach
  Every new situation is modeled by a new set of axioms
- Definitional approach
  Have a fixed logic, and only use definitions to capture new situations

# Proof Assistants: Who's Guarding the Guardians?

Well maintained, heavily used proof assistants
not immune to proofs of False

Most vulnerable place: the definitional mechanism Why is that?

Two approaches to theorem proving

- Axiomatic approach – unsafe
  Every new situation is modeled by a new set of axioms
- Definitional approach – safer
  Have a fixed logic, and only use definitions to capture new situations

# Proof Assistants: Who's Guarding the Guardians?

Well maintained, heavily used proof assistants
not immune to proofs of False

Most vulnerable place: the definitional mechanism Why is that?

Two approaches to theorem proving

- Axiomatic approach – unsafe
  Every new situation is modeled by a new set of axioms
- Definitional approach – safer
  Have a fixed logic, and only use definitions to capture new situations

Definitions can be compiled away without loss of provability

# Proof Assistants: Who's Guarding the Guardians?

Well maintained, heavily used proof assistants
not immune to proofs of False

Most vulnerable place: the definitional mechanism Why is that?

Two approaches to theorem proving

- Axiomatic approach – unsafe
  Every new situation is modeled by a new set of axioms
- Definitional approach – safer but also less convenient
  Have a fixed logic, and only use definitions to capture new situations

Definitions can be compiled away without loss of provability

# Proof Assistants: Who's Guarding the Guardians?

Well maintained, heavily used proof assistants
not immune to proofs of False

Most vulnerable place: the definitional mechanism Why is that?

Two approaches to theorem proving

- Axiomatic approach – unsafe
  Every new situation is modeled by a new set of axioms
- Definitional approach – safer but also less convenient
  Have a fixed logic, and only use definitions to capture new situations

Definitions can be compiled away without loss of provability

Most successful proof assistants

- Enforce the definitional approach
- Strive to make definitions as convenient/expressive as possible

# Proof Assistants: Who's Guarding the Guardians?

Well maintained, heavily used proof assistants
not immune to proofs of False

Most vulnerable place: the definitional mechanism Why is that?

Two approaches to theorem proving

- Axiomatic approach – unsafe
  Every new situation is modeled by a new set of axioms
- Definitional approach – safer but also less convenient
  Have a fixed logic, and only use definitions to capture new situations

Definitions can be compiled away without loss of provability

Most successful proof assistants

- Enforce the definitional approach
- Strive to make definitions as convenient/expressive as possible

$$\begin{aligned} &\text{fact} : \text{Nat} \to \text{Nat} \\ &\qquad \text{fact}\,(0) \quad = \quad 1 \\ &\text{fact}\,(n+1) \quad = \quad (n+1) * \text{fact}\,(n) \end{aligned}$$

# Proof Assistants: Who's Guarding the Guardians?

Well maintained, heavily used proof assistants
not immune to proofs of False

Most vulnerable place: the definitional mechanism Why is that?

Two approaches to theorem proving

- Axiomatic approach – unsafe
  Every new situation is modeled by a new set of axioms
- Definitional approach – safer but also less convenient
  Have a fixed logic, and only use definitions to capture new situations

Definitions can be compiled away without loss of provability

Most successful proof assistants

- Enforce the definitional approach
- Strive to make definitions as convenient/expressive as possible

  $\text{collatz} : \text{Nat} \to \text{LazyList(Nat)}$

$$\text{collatz}\,(n) = \begin{cases} [] & \text{if } n \leq 1 \\ \text{collatz}\,(n \,/\, 2) & \text{if } n > 1 \text{ and } n \text{ even} \\ n :: \text{collatz}\,(3 * n + 1) & \text{if } n > 1 \text{ and } n \text{ odd} \end{cases}$$

# Proof Assistants: Who's Guarding the Guardians?

Well maintained, heavily used proof assistants
not immune to proofs of False

Most vulnerable place: the definitional mechanism Why is that?

Two approaches to theorem proving

- Axiomatic approach – unsafe
  Every new situation is modeled by a new set of axioms
- Definitional approach – safer but also less convenient
  Have a fixed logic, and only use definitions to capture new situations

Definitions can be compiled away without loss of provability

Most successful proof assistants

- Enforce the definitional approach
- Strive to make definitions as convenient/expressive as possible

Proofs of False are a pest feeding on
a proof assistant's definitional ambition

# Proof Assistants: Who's Guarding the Guardians?

Well maintained, heavily used proof assistants
not immune to proofs of False

Proofs of False are a natural consequence of
a proof assistant's definitional ambition

What to do?

# Proof Assistants: Who's Guarding the Guardians?

Well maintained, heavily used proof assistants
not immune to proofs of False

Proofs of False are a natural consequence of
a proof assistant's definitional ambition

What to do?

Formalize and verify

- not their implementation – it evolves too fast


- but their precise logical system – it also evolves, but not so fast
  Pay particular attention to the definitional mechanisms

# Proof Assistants: Who's Guarding the Guardians?

Well maintained, heavily used proof assistants
not immune to proofs of False

Proofs of False are a natural consequence of
a proof assistant's definitional ambition

What to do?

Formalize and verify

- not their implementation – it evolves too fast


- but their precise logical system – it also evolves, but not so fast
  Pay particular attention to the definitional mechanisms

Properties to prove

- those deemed important by users, designers and developers
- including, of course, consistency

# Proof Assistants: Definitional Ambitions

Type Theory (Agda, Coq, Lean, Matita)

- Built-in recursion for types and functions

datatype $\alpha$ list = Nil | Cons $\alpha$ ($\alpha$ list)
fun length xs = case xs of Nil $\Rightarrow$ 0 | Cons x ys $\Rightarrow$ 1 + length ys

# Proof Assistants: Definitional Ambitions

Type Theory (Agda, Coq, Lean, Matita)

- Built-in recursion for types and functions

  datatype $\alpha$ list = Nil | Cons $\alpha$ ($\alpha$ list)
  fun length xs = case xs of Nil $\Rightarrow$ 0 | Cons x ys $\Rightarrow$ 1 + length ys

- Consistency via positivity and guardedness or type-based analysis

# Proof Assistants: Definitional Ambitions

Type Theory (Agda, Coq, Lean, Matita)

- Built-in recursion for types and functions

  datatype $\alpha$ list = Nil | Cons $\alpha$ ($\alpha$ list)
  fun length xs = case xs of Nil $\Rightarrow$ 0 | Cons x ys $\Rightarrow$ 1 + length ys

- Consistency via positivity and guardedness or type-based analysis
  When analysis is flawed $\longrightarrow$ Proofs of False

# Proof Assistants: Definitional Ambitions

Type Theory (Agda, Coq, Lean, Matita)

- Built-in recursion for types and functions

  datatype $\alpha$ list = Nil | Cons $\alpha$ ($\alpha$ list)
  fun length xs = case xs of Nil $\Rightarrow$ 0 | Cons x ys $\Rightarrow$ 1 + length ys

- Consistency via positivity and guardedness or type-based analysis
  When analysis is flawed $\longrightarrow$ Proofs of False

HOL (HOL4, HOL Light, Isabelle/HOL, HOL ProofPower, HOL Zero)

- All recursion reduced to non-recursive definitions
  ($\alpha$, $\beta$) F = unit + $\alpha \times \beta$ $\qquad$ $\alpha$ list = the initial algebra of ($\alpha$, _) F

- No worry about inconsistency through recursion

# Proof Assistants: Definitional Ambitions

Type Theory (Agda, Coq, Lean, Matita)

- Built-in recursion for types and functions

  datatype $\alpha$ list = Nil | Cons $\alpha$ ($\alpha$ list)
  fun length xs = case xs of Nil $\Rightarrow$ 0 | Cons x ys $\Rightarrow$ 1 + length ys

- Consistency via positivity and guardedness or type-based analysis
  When analysis is flawed $\longrightarrow$ Proofs of False

HOL (HOL4, HOL Light, Isabelle/HOL, HOL ProofPower, HOL Zero)

- All recursion reduced to non-recursive definitions
  ($\alpha, \beta$) F = unit + $\alpha \times \beta$ $\qquad$ $\alpha$ list = the initial algebra of ($\alpha$, _) F
- No worry about inconsistency through recursion
- How about the non-recursive definitions?

# Proof Assistants: Definitional Ambitions

Type Theory (Agda, Coq, Lean, Matita)

- Built-in recursion for types and functions

  datatype $\alpha$ list = Nil | Cons $\alpha$ ($\alpha$ list)
  fun length xs = case xs of Nil $\Rightarrow$ 0 | Cons x ys $\Rightarrow$ 1 + length ys

- Consistency via positivity and guardedness or type-based analysis
  When analysis is flawed $\longrightarrow$ Proofs of False

HOL (HOL4, HOL Light, Isabelle/HOL, HOL ProofPower, HOL Zero)

- All recursion reduced to non-recursive definitions
  ($\alpha, \beta$) F = unit + $\alpha \times \beta$ $\qquad$ $\alpha$ list = the initial algebra of ($\alpha$, _) F
- No worry about inconsistency through recursion
- How about the non-recursive definitions? Well...

# Proof Assistants: Definitional Ambitions

Type Theory (Agda, Coq, Lean, Matita)

- Built-in recursion for types and functions

  datatype $\alpha$ list = Nil | Cons $\alpha$ ($\alpha$ list)
  fun length xs = case xs of Nil $\Rightarrow$ 0 | Cons x ys $\Rightarrow$ 1 + length ys

- Consistency via positivity and guardedness or type-based analysis
  When analysis is flawed $\longrightarrow$ Proofs of False

HOL (HOL4, HOL Light, Isabelle/HOL, HOL ProofPower, HOL Zero)

- All recursion reduced to non-recursive definitions
  ($\alpha, \beta$) F = unit + $\alpha \times \beta$ $\qquad$ $\alpha$ list = the initial algebra of ($\alpha$, _) F
- No worry about inconsistency through recursion
- How about the non-recursive definitions? Well...
  - HOL type definitions are not even definitions!
  - In Isabelle/HOL, constant definitions are not entirely non-recursive.

# Part II. Background on HOL

# The HOL Logic

Rank-1 Polymorphic Classical Higher-Order Logic with Choice and Infinity

# The HOL Logic

Rank-1 Polymorphic Classical Higher-Order Logic with Choice and Infinity

We fix:

- a set $K$ of **type constructors**, including bool, ind, $\to$
- a function arOf : $K \to \mathbb{N}$

Types: $\sigma \ ::= \ \alpha \mid (\sigma_1, \ldots, \sigma_{\mathsf{arOf}(k)})\, k$

- a set Const of **constants**, including $=$, $\varepsilon$
- a function tpOf : Const $\to$ Type

$$\begin{array}{rcl} \mathsf{tpOf}(=) & = & \alpha \to \alpha \to \mathsf{bool} \\ \mathsf{tpOf}(\varepsilon) & = & (\alpha \to \mathsf{bool}) \to \alpha \end{array}$$

Terms: The $\lambda$-calculus terms over variables and constants

Typing $t : \sigma$ defined as expected, e.g., $(\lambda x_{\mathsf{bool}}.\ x) : \mathsf{bool} \to \mathsf{bool}$

All other connectives and quantifiers defined from the above, e.g.,

$\mathsf{True} \ \equiv \ (\lambda x_{\mathsf{bool}}.\ x) = (\lambda x_{\mathsf{bool}}.\ x) \qquad \forall x_\sigma.\ \varphi \ \equiv \ (\lambda x_\sigma.\ \varphi) = (\lambda x_\sigma.\ \mathsf{True})$

# The HOL Logic

Formulas: Terms of type bool

Axioms: Equality Axioms, Infinity and Choice

Deduction system: $D$ set of definitions, $\Gamma$ proof context

$$\frac{}{D; \Gamma \vdash \varphi}\ [\varphi \in \text{Axioms} \cup D]\ \text{(Fact)} \qquad \frac{}{D; \Gamma \vdash \varphi}\ [\varphi \in \Gamma]\ \text{(Assum)}$$

$$\frac{D; \Gamma \vdash \varphi}{D; \Gamma \vdash \varphi[\sigma/\alpha]}\ [\alpha \notin \Gamma]\ \text{(T-Inst)} \qquad \frac{D; \Gamma \vdash \varphi}{D; \Gamma \vdash \varphi[t/x_\sigma]}\ [x_\sigma \notin \Gamma]\ \text{(Inst)}$$

$$\frac{}{D; \Gamma \vdash (\lambda x_\sigma.\ t)\ s = t[s/x_\sigma]}\ \text{(Beta)} \qquad \frac{D; \Gamma \vdash \varphi \longrightarrow \chi \qquad D; \Gamma \vdash \varphi}{D; \Gamma \vdash \chi}\ \text{(MP)}$$

$$\frac{D; \Gamma \cup \{\varphi\} \vdash \chi}{D; \Gamma \vdash \varphi \longrightarrow \chi}\ \text{(Imp\_Intro)} \qquad \frac{D; \Gamma \vdash f\ x_\sigma = g\ x_\sigma}{D; \Gamma \vdash f = g}\ [x_\sigma \notin \Gamma]\ \text{(Ext)}$$

Derived rules, e.g.: $\dfrac{D; \Gamma \vdash \varphi}{D; \Gamma \vdash \forall x_\sigma.\ \varphi}\ [x_\sigma \notin \Gamma]\ \text{(All\_Intro)}$

# The HOL Logic: Definitional mechanisms

Constant definitions $c \equiv t$ where

$t : \sigma$ such that TypeVars($t$) $\subseteq$ TypeVars($\sigma$)

$c$ fresh constant (receiving the type $\sigma$)

Type definitions (typedefs) $\tau \equiv t$ where

- $\tau$ has the form $(\alpha_1, \ldots, \alpha_n)\, k$, with $k$ fresh $n$-ary type constructor
- $t : \sigma \to$ bool
- TypeVars($\sigma$) $\subseteq \{\alpha_1, \ldots, \alpha_n\}$

Intuition: $\tau$ is a copy of the subset of $\sigma$ corresponding to $t$, i.e., of $\{x : \sigma \mid t\ x\}$.

Constant definitions are mere equalities, whereas typedefs are formulas stating an isomorphism:

$\tau \equiv t$ is $\exists rep_{\tau\to\sigma}.\ \exists abs_{\sigma\to\tau}.(\tau \approx t)^{abs}_{rep}$

and are accepted only if the predicate is proved non-empty: $\exists x_\sigma.\ t\ x$

HOL keeps its types nonempty

# Part III. Background on Isabelle/HOL

# Isabelle/HOL: A Definitionally More Ambitious HOL

HOL constant definitions: $c \equiv t$

$t : \sigma$

$c$ fresh constant

# Isabelle/HOL: A Definitionally More Ambitious HOL

Isabelle/HOL constant definitions: $c_\sigma \equiv t$

$t : \sigma \leq \mathsf{tpOf}(c)$

$c$ ~~fresh~~ constant

# Isabelle/HOL: A Definitionally More Ambitious HOL

Isabelle/HOL constant definitions: $c_\sigma \equiv t$

$t : \sigma \leq \mathsf{tpOf}(c)$

$c$ ~~fresh~~ constant

HOL: Either only declare a constant, or define it at most general type

Isabelle/HOL: Ad hoc overloading is allowed: can declare a constant, and then define different instances

Haskell-style type classes

# Ad Hoc Overloading in Isabelle/HOL

Example: The class "type endowed with a zero element"

Declare a constant $0 : \alpha$
Define other types and functions that operate on this type class
Define various instances:

- $0 : \text{nat} \equiv$ the number 0
- $0 : \alpha\ \text{list} \equiv [0 : \alpha]$ – a form of recursion

# Ad Hoc Overloading in Isabelle/HOL

Example: The class "type endowed with a zero element"

Declare a constant $0 : \alpha$
Define other types and functions that operate on this type class
Define various instances:

- $0 : \mathsf{nat} \equiv$ the number 0
- $0 : \alpha\ \mathsf{list} \equiv [0 : \alpha]$ – a form of recursion

Safety measures taken against evil overloading

- **Defined instances must be orthogonal**: cannot define both $0_{\alpha \to \mathsf{nat}}$ and $0_{\mathsf{nat} \to \alpha}$
- **Cyclic definitions are not allowed**: cannot define $c_{\mathsf{nat}}$ using $c_{\mathsf{int}}$ and then $c_{\mathsf{int}}$ using $c_{\mathsf{nat}}$

But, until c. 2015:

- No rigorous proof that these measures guarantee consistency.
- Various logical flaws have been discovered and fixed.

# Part IV. Let's Get More Precise

# Definitional Theory

**In Standard HOL:**

Sequence $def_1, \ldots, def_n$ where each $def_i$ is

- either a constant definition $c_\sigma \equiv t$ with $t : \sigma$
  where $c$ is fresh for the signature of $def_1, \ldots, def_{i-1}$
  ($\sigma$ becomes the type of $c$)
- or a typedef $(\alpha_1, \ldots, \alpha_n)\, k \equiv t$ with $t : \sigma \to$ bool
  (postulating the existence of a bijection between
  $(\alpha_1, \ldots, \alpha_n)\, k$ and and $\{x : \sigma \mid t\ x\}$)
  where $k$ is fresh for the signature of $def_1, \ldots, def_{i-1}$

Note: Signatures can also contain declared only constants/types.

# Consistency of Standard-HOL Definitional Theories

**Theorem (Andrew Pitts)**. Any standard-HOL definitional theory $D = \{u_1 \equiv t_1, \ldots, u_n \equiv t_n\}$ is consistent.

Semantic proof: Build standard model satisfied by $D$ in a large enough and closed enough universe $\mathcal{U}$, where:

- Type constructors $k$ of arity $n$ interpreted as functions $[k] : \mathcal{U}^n \to U$
- Types $\sigma$ interpreted as functions $[\sigma] : \mathcal{U}^{\mathsf{TVar}} \to \mathcal{U}$ – their interpretations built from those of their occurring type constructors
- Constants $c : \sigma$ interpreted as families $[c] \in \prod_{\xi \in \mathcal{U}^{\mathbf{TVar}}} [\sigma]\, \xi$
- Term (in particular, formula) interpretations built form those of their occurring constants

By freshness, LHS's of definitions can be interpreted in order:
When interpreting $u_i$, we already know the interpretation of $t_i$.
(For typedefs, we need that $\mathcal{U}$ is closed under taking subsets.)

# Definitional Theory in Isabelle/HOL?

**In Standard HOL:**

Sequence $def_1, \ldots, def_n$ where each $def_i$ is

- Either a constant definition $c_\sigma \equiv t$ with $t : \sigma$
  where $c$ is fresh for the signature of $def_1, \ldots, def_{i-1}$
  ($\sigma$ becomes the type of $c$)
- or a typedef $(\alpha_1, \ldots, \alpha_n)\, k \equiv t$ with $t : \sigma \to$ bool
  (postulating the existence of a bijection between
  $(\alpha_1, \ldots, \alpha_n)\, k$ and and $\{x : \sigma \mid t\ x\}$)
  where $k$ is fresh for the signature of $def_1, \ldots, def_{i-1}$

Note: Signatures can also contain declared only constants/types.

# Definitional Theory in Isabelle/HOL?

**In Standard HOL:**

Sequence $def_1, \ldots, def_n$ where each $def_i$ is

- Either a constant definition $c_\sigma \equiv t$ with $t : \sigma$
  where $c$ is fresh for the signature of $def_1, \ldots, def_{i-1}$
  ($\sigma$ becomes the type of $c$)
- or a typedef $(\alpha_1, \ldots, \alpha_n)\, k \equiv t$ with $t : \sigma \to \text{bool}$
  (postulating the existence of a bijection between
  $(\alpha_1, \ldots, \alpha_n)\, k$ and and $\{x : \sigma \mid t\ x\}$)
  where $k$ is fresh for the signature of $def_1, \ldots, def_{i-1}$

Note: Signatures can also contain declared only constants/types.

**In Isabelle/HOL:**

Constant definitions $c_\sigma \equiv t$ not required to have $c$ fresh; $c$ can be a previously declared constant $c : \tau$ where $\tau \geq \sigma$.

Conditions laxer than freshness are imposed.

# Example: Defining Nominal Logic's Permutation Actions

Declare $\bullet : \mathsf{prm} \to \alpha \to \alpha$

Then define

$$\bullet_{\mathsf{prm}\to\mathsf{atom}\to\mathsf{atom}} \equiv \lambda\pi\ a.\ \mathsf{apply}\ \pi\ a$$

$$\bullet_{\mathsf{prm}\to\mathsf{nat}\to\mathsf{nat}} \equiv \lambda\pi\ n.\ n$$

$$\bullet_{\mathsf{prm}\to\alpha\ \mathsf{list}\to\alpha\ \mathsf{list}} \equiv \lambda\pi\ xs.\ \mathsf{map}\ (\lambda x.\ \pi \bullet x)\ xs$$

$$\bullet_{\mathsf{prm}\to(\alpha\to\beta)\to(\alpha\to\beta)} \equiv \lambda\pi\ f\ x.\ \pi \bullet (f\ (\pi^{-1} \bullet x))$$

# Example: Defining Nominal Logic's Permutation Actions

Declare $\bullet : \mathsf{prm} \to \alpha \to \alpha$

Other type or constant definitions can occur in between... perhaps depending on $\bullet$ instances, and on which $\bullet$ instances may depend on.

Then define

$$\bullet_{\mathsf{prm}\to\mathsf{atom}\to\mathsf{atom}} \equiv \lambda\pi\ a.\ \mathsf{apply}\ \pi\ a$$

$$\bullet_{\mathsf{prm}\to\mathsf{nat}\to\mathsf{nat}} \equiv \lambda\pi\ n.\ n$$

$$\bullet_{\mathsf{prm}\to\alpha\ \mathsf{list}\to\alpha\ \mathsf{list}} \equiv \lambda\pi\ xs.\ \mathsf{map}\ (\lambda x.\ \pi \bullet x)\ xs$$

$$\bullet_{\mathsf{prm}\to(\alpha\to\beta)\to(\alpha\to\beta)} \equiv \lambda\pi\ f\ x.\ \pi \bullet (f\ (\pi^{-1} \bullet x))$$

Also, note the recursive dependencies:

$\bullet_{\mathsf{prm}\to\alpha\ \mathsf{list}\to\alpha\ \mathsf{list}} \rightsquigarrow \bullet_{\mathsf{prm}\to\alpha\to\alpha}$

$\bullet_{\mathsf{prm}\to(\alpha\to\beta)\to(\alpha\to\beta)} \rightsquigarrow \bullet_{\mathsf{prm}\to\alpha\to\alpha}, \bullet_{\mathsf{prm}\to\beta\to\beta}$

# Example: Defining Nominal Logic's Permutation Actions

Declare $\bullet : \mathsf{prm} \to \alpha \to \alpha$

Other type or constant definitions can occur in between... perhaps depending on $\bullet$ instances, and on which $\bullet$ instances may depend on.

Then define

$$\bullet_{\mathsf{prm}\to\mathsf{atom}\to\mathsf{atom}} \equiv \lambda\pi\ a.\ \mathsf{apply}\ \pi\ a$$

$$\bullet_{\mathsf{prm}\to\mathsf{nat}\to\mathsf{nat}} \equiv \lambda\pi\ n.\ n$$

$$\bullet_{\mathsf{prm}\to\alpha\ \mathsf{list}\to\alpha\ \mathsf{list}} \equiv \lambda\pi\ xs.\ \mathsf{map}\ (\lambda x.\ \pi \bullet x)\ xs$$

$$\bullet_{\mathsf{prm}\to(\alpha\to\beta)\to(\alpha\to\beta)} \equiv \lambda\pi\ f\ x.\ \pi \bullet (f\ (\pi^{-1} \bullet x))$$

Also, note the recursive dependencies:

$\bullet_{\mathsf{prm}\to\alpha\ \mathsf{list}\to\alpha\ \mathsf{list}} \rightsquigarrow \bullet_{\mathsf{prm}\to\alpha\to\alpha}$

$\bullet_{\mathsf{prm}\to(\alpha\to\beta)\to(\alpha\to\beta)} \rightsquigarrow \bullet_{\mathsf{prm}\to\alpha\to\alpha}, \bullet_{\mathsf{prm}\to\beta\to\beta}$

But what exactly is a dependency?

# Definitional Dependency Relation – Preliminaries

Built-in type constructors: ind, bool (nullary), $\rightarrow$(binary).
Non-built-in type constructors: all the others, i.e., the defined and declared ones.

Built-in types: Types that have built-in type constructor at the top. E.g., $\alpha \rightarrow$ (ind list).
Hence, non-built-in types are the type variables and those with non-built-in type constructor at the top. E.g., ($\alpha \rightarrow$ ind)list.

Built-in constants: $= : \alpha \rightarrow \alpha \rightarrow$ bool and $\epsilon : (\alpha \rightarrow \text{bool}) \rightarrow \alpha$.

# Definitional Dependency Relation – Preliminaries

For $\sigma \in$ Type, NbiTps($\sigma$) is the set of non-built-in types in $\sigma$:

$\mathsf{NbiTps}(\alpha) = \{\alpha\}$ $\quad \mathsf{NbiTps}(\mathsf{bool}) = \emptyset$ $\quad \mathsf{NbiTps}(\mathsf{ind}) = \emptyset$

$\mathsf{NbiTps}(\sigma_1 \to \sigma_2) = \mathsf{NbiTps}(\sigma_1) \cup \mathsf{NbiTps}(\sigma_2)$

$\mathsf{NbiTps}((\sigma_1, \ldots, \sigma_n)\ k) = \{(\sigma_1, \ldots, \sigma_n)\ k\} \cup \bigcup_{i=1}^{n} \mathsf{NbiTps}(\sigma_i)$
when $k$ is non-built-in type constructor

Similarly, for $t \in$ Term,

NbiTps($t$) is the set of non-built-in types in $t$

NbiCinsts($t$) is the set of non-built-in constant instances in $t$

When tracking definitional dependency of types and terms, we only care about the non-built-in types that they contain.

# Definitional Dependency Relation

Fix definitional theory $D$.
Define dependency relation $\rightsquigarrow$ on $\mathsf{Type} \dot{\cup} \mathsf{CInst}$:

$$u \rightsquigarrow v$$
iff
there is a definition $u \equiv t$ in $D$ s.t. $v \in \mathsf{NbiTps}(t) \dot{\cup} \mathsf{NbiCinsts}(t)$

Definitions can be polymorphic, hence dangerous dependencies can occur through their instances.
To account for this, we consider $\rightsquigarrow$'s type-substitutive closure $\rightsquigarrow^{\downarrow}$:
It relates $u[\rho]$ with $v[\rho]$ when $u \rightsquigarrow v$ and $\rho$ is a type substitution.

Isabelle/HOL has syntactic checks that ensure:

- Orthogonality: For any two definitions $u \equiv t$ and $u' \equiv t'$ in $D$, $u$ and $u'$ are orthogonal (have no common instance)
- Termination of $\rightsquigarrow^{\downarrow}$

Is this enough to prove consistency?

# Parenthesis: Preventing Evil Dependencies Through Types

$$\mathsf{c} : \alpha$$

$$\tau \equiv \{\mathsf{True}, \mathsf{c}_{\mathsf{bool}}\}$$

$$\mathsf{c}_{\mathsf{bool}} \equiv \neg\,(\forall x_\tau\; y_\tau.\; x = y)$$

$$\mathsf{c}_{\mathsf{bool}} \rightsquigarrow^{\downarrow} \tau \rightsquigarrow^{\downarrow} \mathsf{c}_{\mathsf{bool}}$$

Rejected by Isabelle starting 2016, but not by previous versions.

Why has previous work failed to catch this?

- Wenzel 1997
- Haftmann&Wenzel 2006
- Obua 2006 (detailed proof of consistency!)

# Parenthesis: Preventing Evil Dependencies Through Types

$$\mathsf{c} : \alpha$$

$$\tau \equiv \{\mathsf{True}, \mathsf{c}_{\mathsf{bool}}\}$$

$$\mathsf{c}_{\mathsf{bool}} \equiv \neg\,(\forall x_\tau\; y_\tau.\; x = y)$$

$$\mathsf{c}_{\mathsf{bool}} \rightsquigarrow^{\downarrow} \tau \rightsquigarrow^{\downarrow} \mathsf{c}_{\mathsf{bool}}$$

Rejected by Isabelle starting 2016, but not by previous versions.

Why has previous work failed to catch this?

- Wenzel 1997
- Haftmann&Wenzel 2006
- Obua 2006 (detailed proof of consistency!)

They checked dependencies through constant, but not through type definitions – all interesting information about a newly defined type must flow through the embedding-projection pair to its host type, i.e., through defined constants. All, but (finite) cardinality info...

# Consistency of (Isabelle/HOL Checked) Definitional Theories

Let's try a semantic proof *à la* Pitts: Interpret syntax in universe $\mathcal{U}$.

Syntax

$\bullet$ : prm $\to \alpha \to \alpha$

$\bullet_{\mathsf{atom}} \equiv \lambda\pi\ a.\ \mathsf{apply}\ \pi\ a$

$\bullet_{\mathsf{nat}} \equiv \lambda\pi\ n.\ n$

$\bullet_{\alpha\ \mathsf{list}} \equiv \lambda\pi\ xs.\ \mathsf{map}\ (\lambda x.\ \pi \bullet x)\ xs$

$\bullet_{\alpha\to\beta} \equiv \lambda\pi\ f\ x.\ \pi \bullet (f\ (\pi^{-1} \bullet x))$

Semantics

$[\bullet] \in \prod_{A\in\mathcal{U}} \left(A^A\right)^{[\mathsf{prm}]}$

$[\bullet]_{[\mathsf{atom}]} = \Lambda\pi\ a.\ [\mathsf{apply}]\ \pi\ a$

$[\bullet]_{[\mathsf{nat}]} = \Lambda\pi\ a.\ n$

$[\bullet]_{[\mathsf{list}](A)} = \Lambda\pi\ xs.\ [\mathsf{map}]_A\ ([\bullet]_A\ \pi)\ xs$
for all $A \in \mathcal{U}$

$[\bullet]_{A\to B} = \Lambda\pi\ f\ x.\ [\bullet]_B\ \pi\ (f\ ([\bullet]_A\ ([\mathsf{inv}]\ \pi)\ x))$
for all $A, B \in \mathcal{U}$

# Consistency of (Isabelle/HOL Checked) Definitional Theories

Let's try a semantic proof *à la* Pitts: Interpret syntax in universe $\mathcal{U}$.

Syntax

$\bullet : \text{prm} \to \alpha \to \alpha$

$\bullet_{\text{atom}} \equiv \lambda\pi\ a.\ \text{apply}\ \pi\ a$

$\bullet_{\text{nat}} \equiv \lambda\pi\ n.\ n$

$\bullet_{\alpha\ \text{list}} \equiv \lambda\pi\ xs.\ \text{map}\ (\lambda x.\ \pi \bullet x)\ xs$

$\bullet_{\alpha\to\beta} \equiv \lambda\pi\ f\ x.\ \pi \bullet (f\ (\pi^{-1} \bullet x))$

Semantics

$[\bullet] \in \prod_{A\in\mathcal{U}} (A^A)^{[\text{prm}]}$

$[\bullet]_{[\text{atom}]} = \Lambda\pi\ a.\ [\text{apply}]\ \pi\ a$

$[\bullet]_{[\text{nat}]} = \Lambda\pi\ a.\ n$

$[\bullet]_{[\text{list}](A)} = \Lambda\pi\ xs.\ [\text{map}]_A\ ([\bullet]_A\ \pi)\ xs$
for all $A \in \mathcal{U}$

$[\bullet]_{A\to B} = \Lambda\pi\ f\ x.\ [\bullet]_B\ \pi\ (f\ ([\bullet]_A\ ([\text{inv}]\ \pi)\ x))$
for all $A, B \in \mathcal{U}$

Must define a family of functions $([\bullet]_A)_{A\in\mathcal{U}}$ via some equalities.
All we know: Syntactic counterparts orthogonal and terminating.
Are the semantic definitions:

- Orthogonal? Can make them, with set-theoretic acrobatics.
- Terminating? Not even clear what this means.

How to connect semantic and syntactic termination?

# Solution: Rethinking the Interpretation of Polymorphism

Interpret $\bullet$ : prm $\to \alpha \to \alpha$

not as $[\bullet] \in \prod_{A \in \mathcal{U}} \left(A^A\right)^{[\mathsf{prm}]}$

but as $[\bullet] \in \prod_{\sigma \in \mathsf{GroundType}} \left([\sigma]^{[\sigma]}\right)^{[\mathsf{prm}]}$

Blend of syntax and semantics:

- polymorphism interpreted syntactically
- everything else interpreted semantically

# Solution: Rethinking the Interpretation of Polymorphism

For the polymorphic type $\alpha \to \text{bool}$:

- no "global" interpretation $\prod_{A \in \mathcal{U}} A^{[\text{bool}]}$
- but only interpretations of its ground-instances, $\sigma \to \text{bool}$ for $\sigma \in \text{GroundType}$, as $[\sigma]^{[\text{bool}]}$

For a polymorphic constant $c : \alpha \to \text{bool}$:

- no "global" interpretation in $\prod_{A \in \mathcal{U}} A^{[\text{bool}]}$
- but only interpretations of its ground instances $c_\sigma$ for $\sigma \in \text{GroundType}$, as an element of $[\sigma]^{[\text{bool}]}$

The (truth of the) formula $\forall x : \alpha.\ c_\alpha\ x$

- no longer means: for all $A \in \mathcal{U}$ and $x \in A$, $[c]_A\ x = \text{True}$
- but rather: for all $\sigma \in \text{GroundType}$ and $x \in [\sigma]$, $[c_\sigma]x = \text{True}$

# Solution: Rethinking the Interpretation of Polymorphism

**Lemma.** The HOL rules are sound for any ground interpretation.

**Theorem.** Any Isabelle/HOL-checked definitional theory $D$ is true in some ground interpretation, hence is consistent.

Proof idea. The desired interpretation is defined by well-founded recursion w.r.t. $\rightsquigarrow^{\downarrow}$. For any instance $(u[\rho], t[\rho])$ of a definition $u \equiv t$ in $D$:

- All non-built-in items in $t[\rho]$ are "smaller" than $u[\rho]$ – hence have already been interpreted at the time we interpret $u[\rho]$ according to the definition's semantics.

- All built items have a fixed standard interpretation.

(Uses partial interpretations of well-behaved signature fragments.)

O.K. & A.P. A Consistent Foundation for Isabelle/HOL. ITP'15.

# Ground Interpretation Illustrated

Syntax

$\bullet : \mathsf{prm} \to \alpha \to \alpha$

$\bullet_{\alpha\ \mathsf{list}} \equiv \lambda\pi\ xs.\ \mathsf{map}\ (\lambda x.\ \pi \bullet x)\ xs$

$\bullet_{\alpha\to\beta} \equiv \lambda\pi\ f\ x.\ \pi \bullet (f\ (\pi^{-1} \bullet x))$

Pitts Semantics

$[\bullet] \in \prod_{A\in\mathcal{U}} (A^A)^{\mathsf{prm}}$

$[\bullet]_{[\mathsf{list}](A)} = \Lambda\pi\ xs.\ [\mathsf{map}]_A\ ([\bullet]_A\ \pi)\ xs$
for all $A \in \mathcal{U}$

$[\bullet]_{A\to B} = \Lambda\pi\ f\ x.\ [\bullet]_B\ \pi\ (f\ ([\bullet]_A\ ([\mathsf{inv}]\ \pi)\ x))$
for all $A, B \in \mathcal{U}$

Ground Semantics

$[\bullet] \in \prod_{\sigma\in\mathsf{GroundType}} ([\sigma]^{[\sigma]})^{\mathsf{prm}}$

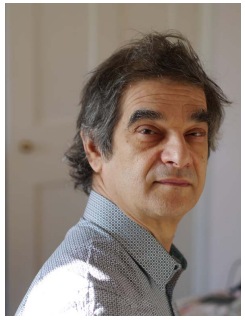
$[\bullet]_{[\sigma\ \mathsf{list}]} = \ldots$
for all $\sigma \in \mathsf{GroundType}$

$[\bullet]_{[\sigma\to\tau]} = \ldots$
for all $\sigma, \tau \in \mathsf{GroundType}$

Difference: Ground semantics equations are a well-formed definition!

$[\bullet]_{[(\mathsf{nat}\to\mathsf{nat})\ \mathsf{list}]}$ defined before $[\bullet]_{[\mathsf{nat}\to\mathsf{nat}]}$ defined before $[\bullet]_{[\mathsf{nat}]}$ etc.

# Reaction to Our Consitency Proof



Larry Paulson

"It's a bit puzzling, not to say worrying, to want a [new] set-theoretic semantics for plain definitions. The point of definitions (and the origin of the idea that they preserve consistency) is that they are abbreviations."

# Towards a Syntactic Proof of Consistency

Main difficulty:

- Types and constant definitions are mutually dependent
- Typedefs cannot be seen as abbreviations

Typedefs create types $\tau$ corresponding to subsets of existing types $\sigma$

$$\exists rep_{\tau\to\sigma}.\ \exists abs_{\sigma\to\tau}.(\tau \approx t)^{abs}_{rep}$$

$\tau$'s definition cannot be unfolded **inside HOL**

Similarly, there is no $\sqrt{-1}$ **inside** $\mathbb{R}$...

# Towards a Syntactic Proof of Consistency

Main difficulty:

- Types and constant definitions are mutually dependent
- Typedefs cannot be seen as abbreviations

Typedefs create types $\tau$ corresponding to subsets of existing types $\sigma$

$$\exists rep_{\tau\to\sigma}.\ \exists abs_{\sigma\to\tau}.(\tau \approx t)^{abs}_{rep}$$

$\tau$'s definition cannot be unfolded **inside HOL**

Similarly, there is no $\sqrt{-1}$ **inside** $\mathbb{R}$... however, we can complete $\mathbb{R}$ to $\mathbb{C}$, where $\sqrt{-1}$ makes sense.

# HOL with Comprehension Types (HOLC)

Type-fortified logic 


Makes room for unfolding type definitions

$\alpha\ k \equiv \{\!|\lambda x_\alpha.\ c_{\alpha\to\mathsf{bool}}\ d_\alpha|\!\}$
$\mathsf{c}_{\alpha\ k\to\mathsf{bool}} \equiv \lambda x_{\alpha\ k}.\ \ c_{\alpha\to\mathsf{bool}}\ d_\alpha$

# HOL with Comprehension Types (HOLC)

Type-fortified logic 


Makes room for unfolding type definitions

$\alpha\ k \equiv \{\!|\lambda x_\alpha.\ c_{\alpha\to\mathsf{bool}}\ d_\alpha|\!\}$
$\mathsf{c}_{\alpha\ k\to\mathsf{bool}} \equiv \lambda x_{\alpha\ k}.\ c_{\alpha\to\mathsf{bool}}\ d_\alpha$

$(\mathsf{bool}\ k)\ k$

# HOL with Comprehension Types (HOLC)

Type-fortified logic 

Makes room for unfolding type definitions

$\alpha\ k \equiv \{\!|\lambda x_\alpha.\ c_{\alpha\to\mathsf{bool}}\ d_\alpha|\!\}$

$\mathsf{c}_{\alpha\ k\to\mathsf{bool}} \equiv \lambda x_{\alpha\ k}.\ c_{\alpha\to\mathsf{bool}}\ d_\alpha$

$$(\mathsf{bool}\ k)\ k$$
$$\Downarrow$$
$$\{\!|\lambda x_{\mathsf{bool}\ k}.\ c_{\mathsf{bool}\ k\to\mathsf{bool}}\ d_{\mathsf{bool}\ k}|\!\}$$

# HOL with Comprehension Types (HOLC)

Type-fortified logic 

Makes room for unfolding type definitions

$\alpha\ k \equiv \{\!|\lambda x_\alpha.\ c_{\alpha\to\mathsf{bool}}\ d_\alpha|\!\}$

$c_{\alpha\ k\to\mathsf{bool}} \equiv \lambda x_{\alpha\ k}.\ c_{\alpha\to\mathsf{bool}}\ d_\alpha$

$$(\mathsf{bool}\ k)\ k$$
$$\Downarrow$$
$$\{\!|\lambda x_{\mathsf{bool}\ k}.\ c_{\mathsf{bool}\ k\to\mathsf{bool}}\ d_{\mathsf{bool}\ k}|\!\}$$
$$\Downarrow$$
$$\{\!|\lambda x_{\mathsf{bool}\ k}.\ (\lambda x_{\mathsf{bool}\ k}.\ c_{\mathsf{bool}\to\mathsf{bool}}\ d_{\mathsf{bool}})\ d_{\mathsf{bool}\ k}|\!\}$$

# HOL with Comprehension Types (HOLC)

Type-fortified logic 

Makes room for unfolding type definitions

$\alpha\ k \equiv \{\!|\lambda x_\alpha.\ c_{\alpha\to\mathsf{bool}}\ d_\alpha|\!\}$

$c_{\alpha\ k\to\mathsf{bool}} \equiv \lambda x_{\alpha\ k}.\ c_{\alpha\to\mathsf{bool}}\ d_\alpha$

$$(\mathsf{bool}\ k)\ k$$
$$\Downarrow$$
$$\{\!|\lambda x_{\mathsf{bool}\ k}.\ c_{\mathsf{bool}\ k\to\mathsf{bool}}\ d_{\mathsf{bool}\ k}|\!\}$$
$$\Downarrow$$
$$\{\!|\lambda x_{\mathsf{bool}\ k}.\ (\lambda x_{\mathsf{bool}\ k}.\ c_{\mathsf{bool}\to\mathsf{bool}}\ d_{\mathsf{bool}})\ d_{\mathsf{bool}\ k}|\!\}$$
$$\Downarrow$$
$$\{\!|\lambda x_{\{\!|\lambda x_{\mathsf{bool}}.\ c_{\mathsf{bool}\to\mathsf{bool}}\ d_{\mathsf{bool}}|\!\}}.(\lambda x_{\{\!|\lambda x_{\mathsf{bool}}.\ c_{\mathsf{bool}\to\mathsf{bool}}\ d_{\mathsf{bool}}|\!\}}.c_{\mathsf{bool}\to\mathsf{bool}}\,d_{\mathsf{bool}})\,d_{\{\!|\lambda x_{\mathsf{bool}}.\ c_{\mathsf{bool}\to\mathsf{bool}}\ d_{\mathsf{bool}}|\!\}}|\!\}$$

# From HOL to HOLC

$$\text{Types:}\quad \sigma = \alpha \mid (\sigma_1, \ldots, \sigma_{\mathsf{arOf}(k)})\, k \mid \{\!|t|\!\}$$

$$\text{Terms:}\quad t = x_\sigma \mid c_\sigma \mid t_1\ t_2 \mid \lambda x_\sigma.\, t$$

for $t : \sigma \to \mathsf{bool}$, $\{\!|t|\!\}$ means $\{x : \sigma \mid t\ x\}$

# From HOL to HOLC

$$\text{Types:}\quad \sigma = \alpha \mid (\sigma_1, \ldots, \sigma_{\mathsf{arOf}(k)})\, k \mid \{\!|t|\!\}$$

$$\text{Terms:}\quad t = x_\sigma \mid c_\sigma \mid t_1\ t_2 \mid \lambda x_\sigma.\, t$$

for $t : \sigma \to \mathsf{bool}$, $\{\!|t|\!\}$ means $\{x : \sigma \mid t\ x\}$

Typing:

$$\frac{\alpha \in \mathsf{TVar}}{\mathsf{wf}(\alpha)}\ (\mathsf{W}_1) \qquad \frac{\mathsf{wf}(\sigma_1)\ \ldots\ \mathsf{wf}(\sigma_{\mathsf{arOf}(k)})}{\mathsf{wf}((\sigma_1, \ldots, \sigma_{\mathsf{arOf}(k)})\, k)}\ (\mathsf{W}_2) \qquad \frac{t : \sigma \to \mathsf{bool}}{\mathsf{wf}(\{\!|t|\!\})}\ (\mathsf{W}_3)$$

$$\frac{t : \tau \quad \mathsf{wf}(\sigma)}{\lambda x_\sigma.\, t : \sigma \Rightarrow \tau}\ (\mathsf{Abs}) \qquad \frac{x \in \mathsf{VarN} \quad \mathsf{wf}(\sigma)}{x_\sigma : \sigma}\ (\mathsf{Var})$$

$$\frac{c \in \mathsf{Const} \quad \mathsf{wf}(\tau) \quad \tau \leq \mathsf{tpOf}(c)}{c_\tau : \tau}\ (\mathsf{Const}) \qquad \frac{t_1 : \sigma \Rightarrow \tau \quad t_2 : \sigma}{t_1\ t_2 : \tau}\ (\mathsf{App})$$

# From HOL to HOLC

Axioms and deduction rules: Same as for HOL, except that:

- Deduction not parameterized by definitional theory $D$

$$\not{D}; \Gamma \vdash \varphi$$

# From HOL to HOLC

Axioms and deduction rules: Same as for HOL, except that:

- Deduction not parameterized by definitional theory $D$

$$\emptyset; \Gamma \vdash \varphi$$

# From HOL to HOLC

Axioms and deduction rules: Same as for HOL, except that:

- Deduction not parameterized by definitional theory $D$

$$\emptyset; \Gamma \vdash \varphi$$

- In exchange, a comprehension axiom is added:

$$\forall t_{\alpha \to \mathsf{bool}}.\ (\exists x_{\alpha}.\ t\ x) \longrightarrow \exists \mathit{rep}_{\{\!|t|\!\}\to\alpha}.\ \exists \mathit{abs}_{\alpha\to\{\!|t|\!\}}.(\{\!|t|\!\} \approx t)^{\mathit{abs}}_{\mathit{rep}}$$

which captures all HOL typedefs

# Proof of Consistency

Fix Isabelle/HOL-checked definitional theory $D$.
Translate HOL types and terms to HOLC by recursive unfolding of $D$, until a normal form is reached. This terminates because the call graph is included in (an expansion of) $\rightsquigarrow^{\downarrow}$.

**Theorem.** $D; \Gamma \vdash \varphi$ in HOL implies $\emptyset; \mathsf{UNF}(\Gamma) \vdash \mathsf{UNF}(\varphi)$ in HOLC

# Proof of Consistency

Fix Isabelle/HOL-checked definitional theory $D$.
Translate HOL types and terms to HOLC by recursive unfolding of $D$, until a normal form is reached. This terminates because the call graph is included in (an expansion of) $\rightsquigarrow^{\downarrow}$.

**Theorem.** $D; \Gamma \vdash \varphi$ in HOL implies $\emptyset; \mathsf{UNF}(\Gamma) \vdash \mathsf{UNF}(\varphi)$ in HOLC

Main difficulty in proof: the type instantiation rule

$$\frac{D; \Gamma \vdash \varphi}{D; \Gamma \vdash \varphi[\sigma/\alpha]}\ [\alpha \notin \Gamma]\ (\mathsf{T\text{-}Inst})$$

UNF does not commute with type substitutions:

$$\mathsf{UNF}(t[\sigma/\alpha]) \neq \mathsf{UNF}(t)[\mathsf{UNF}(\sigma)/\alpha]$$

# Proof of Consistency

Culprit: Of course, ad hoc overloading

Declare $c : \alpha$

Define $c_{\mathsf{nat}} \equiv 0$

$\mathsf{UNF}(c_\alpha[\mathsf{nat}/\alpha]) = 0 \neq c_{\mathsf{nat}} = c_\alpha[\mathsf{nat}/\alpha] = \mathsf{UNF}(c_\alpha)[\mathsf{nat}/\alpha]$

# Proof of Consistency

Culprit: Of course, ad hoc overloading

Declare $c : \alpha$

Define $c_{\mathsf{nat}} \equiv 0$

$\mathrm{UNF}(c_\alpha[\mathsf{nat}/\alpha]) = 0 \neq c_{\mathsf{nat}} = c_\alpha[\mathsf{nat}/\alpha] = \mathrm{UNF}(c_\alpha)[\mathsf{nat}/\alpha]$

Solution:

Refrain from applying (T-Inst) in the middle of proofs, but only apply it at the beginning – OK for rank 1 polymorphism

Hence it suffices that

- $\mathrm{UNF}(\varphi[\sigma/\alpha]) = \mathrm{UNF}(\varphi)[\mathrm{UNF}(\sigma)/\alpha]$ for HOL axioms $\varphi$
- $\mathrm{UNF}(\varphi[\sigma/\alpha])$ is a HOLC tautology for $\varphi \in D$
  - constant definitions become equalities of identical terms
  - typedefs become instances of HOLC's compr. axiom

# Proof of Consistency

**Theorem.** Assume $D$ is an Isabelle/HOL-checked definitional theory. Then $D \vdash \varphi$ in HOL implies $\vdash \mathrm{UNF}(\varphi)$ in HOLC

Have complicated the logic, but have gotten rid of definitions.

# Proof of Consistency

**Theorem.** Assume $D$ is an Isabelle/HOL-checked definitional theory. Then $D \vdash \varphi$ in HOL implies $\vdash \mathrm{UNF}(\varphi)$ in HOLC

Have complicated the logic, but have gotten rid of definitions.

**Theorem (Easy).** HOLC is consistent.

**Corollary.** Isabelle/HOL-checked definitional theories are consistent.

O.K. & A.P. Comprehending Isabelle/HOL's Consistency. ESOP'17.

# Proof of Consistency

**Theorem.** Assume $D$ is an Isabelle/HOL-checked definitional theory. Then $D \vdash \varphi$ in HOL implies $\vdash \mathrm{UNF}(\varphi)$ in HOLC

Have complicated the logic, but have gotten rid of definitions.

Have have gotten rid of definitions, but have complicated the logic.

# Proof of Consistency

**Theorem.** Assume $D$ is an Isabelle/HOL-checked definitional theory. Then $D \vdash \varphi$ in HOL implies $\vdash \mathrm{UNF}(\varphi)$ in HOLC

Have complicated the logic, but have gotten rid of definitions.

Have have gotten rid of definitions, but have complicated the logic.

Can we get rid of definitions while staying in HOL?

# Proof of Consistency

**Theorem.** Assume $D$ is an Isabelle/HOL-checked definitional theory. Then $D \vdash \varphi$ in HOL implies $\vdash \mathrm{UNF}(\varphi)$ in HOLC

Have complicated the logic, but have gotten rid of definitions.

Have have gotten rid of definitions, but have complicated the logic.

Can we get rid of definitions while staying in HOL? Yes, we can.

$\mathsf{Type} \overset{\mathsf{HOST}}{\to} \mathsf{Type}$ $\quad$ $\mathsf{Type} \overset{\mathsf{REL}}{\to} \mathsf{Term}$ $\quad$ $\mathsf{Term} \overset{\mathsf{UNF}}{\to} \mathsf{Term}$

**Theorem.**
(1) $\mathsf{REL}(\sigma) : \mathsf{HOST}(\sigma) \to \mathsf{bool}$ and $\vdash \exists x_{\mathsf{HOST}(\sigma)}.\, \mathsf{REL}(\sigma)\, x$.
(2) $t : \sigma$ implies $\mathsf{UNF}(t) : \mathsf{HOST}(\sigma)$ and $\vdash \mathsf{REL}(\sigma)\, \mathsf{UNF}(t)$.
(3) $D \vdash \varphi$ implies $\vdash \mathsf{UNF}(\varphi)$.

**Corollary.** (Isabelle/)HOL-checked definitional theories are conservative over initial HOL.

O.K. & A.P. Safety and Conservativity for HOL and Isabelle/HOL. POPL'18.

$$\mathsf{HOST}(\alpha) = \alpha$$
$$\mathsf{HOST}((\sigma_1, \ldots, \sigma_m)\ k) = (\mathsf{HOST}(\sigma_1), \ldots, \mathsf{HOST}(\sigma_m))\ k,$$
if $k \in \mathsf{Decl}$
$$\mathsf{HOST}((\sigma_1, \ldots, \sigma_m)\ k) = \mathsf{HOST}(\sigma[\sigma_1/\alpha_1, \ldots, \sigma_m/\alpha_m]),$$
if $(\alpha_1, \ldots, \alpha_m)\ k \equiv t$ is in $D$ and $t : \sigma \Rightarrow \mathsf{bool}$

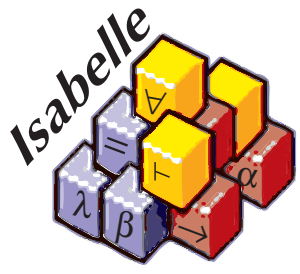
$$\mathsf{REL}(\sigma) = \lambda x_\sigma.\ \mathsf{True}, \quad \text{if } \sigma \in \mathsf{TVar} \cup \{\mathsf{bool}, \mathsf{ind}\}$$
$$\mathsf{REL}(\sigma_1 \Rightarrow \sigma_2) = \lambda f_{\mathsf{HOST}(\sigma_1) \Rightarrow \mathsf{HOST}(\sigma_2)}.\ \forall x_{\mathsf{HOST}(\sigma_1)}.\ \mathsf{REL}(\sigma_2)\ (f\ x)$$
$$\mathsf{REL}((\sigma_1, \ldots, \sigma_m)\ k) = \lambda x_{(\mathsf{HOST}(\sigma_1), \ldots, \mathsf{HOST}(\sigma_m))\ k}.\mathsf{True}, \quad \text{if } k \in \mathsf{Decl}$$
$$\mathsf{REL}((\sigma_1, \ldots, \sigma_m)\ k) = \mathsf{UNF}(t')$$
if $(\alpha_1, \ldots, \alpha_m)\ k \equiv t$ is in $D$,
where $t' = t[\sigma_1/\alpha_1, \ldots, \sigma_m/\alpha_m]$

$$\mathsf{UNF}(x_\sigma) = \mathsf{if\_t\_e}\ (\mathsf{REL}(\sigma)\ x_{\mathsf{HOST}(\sigma)})\ x\ (\varepsilon\ \mathsf{REL}(\sigma))$$
$$\mathsf{UNF}(c_\sigma) = c_{\mathsf{HOST}(\sigma)}, \quad \text{if } c \in \Sigma_{\mathsf{init}}$$
$$\mathsf{UNF}(c_\sigma) = \mathsf{if\_t\_e}\ (\mathsf{REL}(\sigma)\ c_{\mathsf{HOST}(\sigma)})\ c_{\mathsf{HOST}(\sigma)}\ (\varepsilon\ \mathsf{REL}(\sigma)), \quad \text{if } c \in \mathsf{Decl}$$
$$\mathsf{UNF}(c_\sigma) = \mathsf{UNF}(t[\rho]), \quad \text{if } c_\tau \equiv t \text{ is in } D \text{ and } \sigma \leq_\rho \tau$$
$$\mathsf{UNF}(t_1\ t_2) = \mathsf{UNF}(t_1)\ \mathsf{UNF}(t_2)$$
$$\mathsf{UNF}(\lambda x_\sigma.\ t) = \lambda x_{\mathsf{HOST}(\sigma)}.\ \mathsf{UNF}(t)$$

# Epilogue

?

Proof of Consistency




# Conclusion

We gave three different proofs of consistency or stronger properties for Isabelle/HOL, by compiling away the HOL definitions:

- Interpretation in semantic domain
- Translation to HOLC
- Translation to HOL

# Proofs developed rigorously

**Theorem 16** Assume $D$ is a definitional theory. Then it has a total-fragment model, i.e., there exists a $\top$-interpretation $\mathcal{I}$ such that $\mathcal{I} \models D$.

*Proof.* Since $\rightsquigarrow^{\downarrow}$ is terminating, $\rightsquigarrow^{\downarrow+}$ is also terminating. By well-founded recursion and induction on $\rightsquigarrow^{\downarrow+}$, for each $u \in \mathsf{GType}^\bullet \cup \mathsf{GCInst}^\bullet$, we define $[u]$ simultaneously with proving that the following hold:

$\mathsf{A}_u$: $\mathcal{I}_u = (([v])_{v \in T_u}, ([v])_{v \in C_u})$ is an $F_u$-interpretation.[8]
$\mathsf{B}_u$: If $u \in \mathsf{GType}^\bullet$ (i.e., $u$ is a type) then $[u] \neq \emptyset$; and if $u$ has the form $c_\tau$ (i.e., $u$ is a constant) then $[u] \in [\tau]$.

We assume that, for all $v \in \mathsf{GType}^\bullet \cup \mathsf{GCInst}^\bullet$ such that $u \rightsquigarrow^{\downarrow+} v$ holds, $[v]$ has been defined and $\mathsf{A}_v$ and $\mathsf{B}_v$ hold.

We first show that $\mathsf{A}_u$ holds. For this, we first assume $\tau \in T_u$ and need to prove $[\tau] \neq \emptyset$; but this is precisely $\mathsf{B}_{c_\tau}$, which holds by the induction hypothesis, since $u \rightsquigarrow^{\downarrow+} \tau$. Now, we assume $c_\tau \in C_u$ and need to prove $[c_\tau] \in [\tau]$; but this is precisely $\mathsf{B}_{c_\tau}$, which holds by the induction hypothesis, since $u \rightsquigarrow^{\downarrow+} c_\tau$.

Next, we define $[u]$. We say that a definition $w \equiv s$ matches $u$ if there exists a ground type substitution $\theta$ with $u = \theta(w)$. We distinguish the following cases:

1. There exists no definition in $D$ that matches $u$. Here we have two subcases:
   - $u \in \mathsf{GType}^\bullet$. Then we define $[u] = \{*\}$.
   - $u \in \mathsf{GCInst}^\bullet$. Say $u$ has the form $c_\sigma$. By Lemma 14, we have that $c_\sigma \rightsquigarrow^{\downarrow} \tau$ for all $\tau \in \mathsf{types}^\bullet(\sigma)$. Hence $\mathsf{types}^\bullet(\sigma) \subseteq T_u$, which means $\sigma \in \mathsf{Type}^{F_u}$. Therefore, using also $\mathsf{A}_u$, we can speak of the value $[\sigma]^{F_u, \mathcal{I}_u}$ (obtained from the $F_u$-interpretation $\mathcal{I}_u$). We define $[u] = \mathsf{choice}([\sigma]^{F_u, \mathcal{I}_u})$.
2. There exists a definition $w \equiv s$ in $D$ that matches $u$. Then let $\theta$ be such that $u = \theta(w)$, and let $t = \theta(s)$. By Lemma 15(2) we have $t \in \mathsf{Term}^{F_u}$. Therefore, using also $\mathsf{A}_u$, we can speak of the value $[t]^{F_u, \mathcal{I}_u}$. We have two subcases:
   - $u \in \mathsf{GCInst}^\bullet$. Then we define $[u] = [t]^{F_u, \mathcal{I}_u}$.
   - $u \in \mathsf{GType}^\bullet$. Then the type of $t$ has the form $\sigma \to \mathsf{bool}$. And since $\mathsf{types}^\bullet(\sigma) \subseteq \mathsf{types}^\bullet(t) \subseteq T_u$, by Lemma 8(1) we obtain $\sigma \in \mathsf{Cl}(T_u)$, i.e., $\sigma \in \mathsf{Type}^{F_u}$. Therefore, using also $\mathsf{A}_u$, we can speak of the value $[\sigma]^{F_u, \mathcal{I}_u}$. We have two subsubcases:

# Proofs developed rigorously... save for a few routine lemmas.

The operators $\mathsf{consts}^\bullet$ and $\mathsf{types}^\bullet$ commute with type substitutions and behave well w.r.t. free variables, subterms and term substitution.

**Lemma 4** The following hold:
(1) $\mathsf{types}^\bullet(\rho(\tau)) = \{\rho(\sigma) \mid \sigma \in \mathsf{types}^\bullet(\tau)\}$
(2) $\mathsf{types}^\bullet(\rho(t)) = \{\rho(\sigma) \mid \sigma \in \mathsf{types}^\bullet(t)\}$
(3) $\mathsf{consts}^\bullet(\rho(t)) = \{c_{\rho(\sigma)} \mid c_\sigma \in \mathsf{consts}^\bullet(t)\}$
(4) If $x_\sigma \in \mathsf{FV}(t)$, then $\mathsf{types}^\bullet(\sigma) \subseteq \mathsf{types}^\bullet(t)$
(5) If $t'$ is a subterm of $t$, then $\mathsf{types}^\bullet(t') \subseteq \mathsf{types}^\bullet(t)$ and cons$^\bullet(t')$ $\subseteq$ cons$^\bullet(t)$
(6) If $\mathsf{tpOf}(t') = \sigma$, then $\mathsf{types}^\bullet(t[t'/x_\sigma]) \subseteq \mathsf{types}^\bullet(t) \cup \mathsf{types}^\bullet(t')$ and $\mathsf{consts}^\bullet(t[t'/x_\sigma]) \subseteq \mathsf{consts}^\bullet(t) \cup \mathsf{consts}^\bullet(t')$

These are wrong.

*Proof.* By straightforward induction on the type $\tau$ or the term $t$, noticing that the application of a type or term substitution leaves unchanged the top type or term constructor. For example, the non-built-in case in the induction for point (1) goes as follows. Assume $k \neq \mathsf{bool}, \mathsf{ind}, \to$. Then, applying the definitions of substitution and of $\mathsf{types}^\bullet$ and writing $\rho(\overline{\tau})$ for the tuple obtained from $\overline{\tau}$ by applying $\rho$ componentwise, we have:

$$\mathsf{types}^\bullet(\rho(\overline{\tau}\,k)) = \mathsf{types}^\bullet((\rho(\overline{\tau}))\,k) = \mathsf{types}^\bullet(\{(\rho(\overline{\tau}))\,k\}) = \{\rho(\sigma) \mid \sigma \in \mathsf{types}^\bullet(\overline{\tau}\,k)\}$$

□

Thank You

# Proofs developed rigorously... save for a few routine lemmas.

The operators $\mathsf{consts}^\bullet$ and $\mathsf{types}^\bullet$ commute with type substitutions and behave well w.r.t. free variables, subterms and term substitution.

**Lemma 4** The following hold:
(1) $\mathsf{types}^\bullet(\rho(\tau)) = \{\rho(\sigma) \mid \sigma \in \mathsf{types}^\bullet(\tau)\}$
(2) $\mathsf{types}^\bullet(\rho(t)) = \{\rho(\sigma) \mid \sigma \in \mathsf{types}^\bullet(t)\}$
(3) $\mathsf{consts}^\bullet(\rho(t)) = \{c_{\rho(\sigma)} \mid c_\sigma \in \mathsf{consts}^\bullet(t)\}$
(4) If $x_\sigma \in \mathsf{FV}(t)$, then $\mathsf{types}^\bullet(\sigma) \subseteq \mathsf{types}^\bullet(t)$
(5) If $t'$ is a subterm of $t$, then $\mathsf{types}^\bullet(t') \subseteq \mathsf{types}^\bullet(t)$ and cons$^\bullet(t')$ $\subseteq$ cons$^\bullet(t)$
(6) If $\mathsf{tpOf}(t') = \sigma$, then $\mathsf{types}^\bullet(t[t'/x_\sigma]) \subseteq \mathsf{types}^\bullet(t) \cup \mathsf{types}^\bullet(t')$ and $\mathsf{consts}^\bullet(t[t'/x_\sigma]) \subseteq \mathsf{consts}^\bullet(t) \cup \mathsf{consts}^\bullet(t')$

These are wrong.

*Proof.* By straightforward induction on the type $\tau$ or the term $t$, noticing that the application of a type or term substitution leaves unchanged the top type or term constructor. For example, the non-built-in case in the induction for point (1) goes as follows. Assume $k \neq \mathsf{bool}, \mathsf{ind}, \to$. Then, applying the definitions of substitution and of $\mathsf{types}^\bullet$ and writing $\rho(\overline{\tau})$ for the tuple obtained from $\overline{\tau}$ by applying $\rho$ componentwise, we have:

$$\mathsf{types}^\bullet(\rho(\overline{\tau}\,k)) = \mathsf{types}^\bullet((\rho(\overline{\tau}))\,k) = \mathsf{types}^\bullet(\{(\rho(\overline{\tau}))\,k\}) = \{\rho(\sigma) \mid \sigma \in \mathsf{types}^\bullet(\overline{\tau}\,k)\}$$

□

# Conclusion

Three different proofs of consistency, using:

- Interpretation in semantic domain
- Translation to HOLC
- Translation to HOL

Faulty lemma affects all of them.

Their well-definedness needs that a certain relation terminates:

$u \rightsquigarrow^{\Downarrow} v$ iff there exist a definition $u \equiv t$ in $D$ and a type substitution $\rho$ s.t. $v \in \mathsf{NbiTps}\,(t[\rho]) \dot{\cup} \mathsf{NbiCinsts}\,(t[\rho])$

... whereas we know that a different relation terminates:

$u \rightsquigarrow^{\downarrow} v$ iff there exist $u', v', \rho$ and a definition $u' \equiv t$ in $D$ s.t. $u = u'[\rho]$, $v = v'[\rho]$ and $v' \in \mathsf{NbiTps}\,(t) \dot{\cup} \mathsf{NbiCinsts}\,(t)$

Faulty lemma would imply that that $\rightsquigarrow^{\Downarrow} \subseteq \rightsquigarrow^{\downarrow}$.

# Conclusion

Three different proofs of consistency, using:

- Interpretation in semantic domain
- Translation to HOLC
- Translation to HOL

Faulty lemma affects all of them.

Their well-definedness needs that a certain relation terminates:

$u \rightsquigarrow^{\Downarrow} v$ iff there exist a definition $u \equiv t$ in $D$ and a type substitution $\rho$ s.t. $v \in \mathsf{NbiTps}\,(t[\rho]) \dot{\cup} \mathsf{NbiCinsts}\,(t[\rho])$

... whereas we know that a different relation terminates:

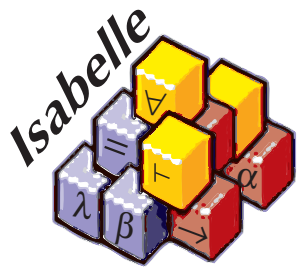
$u \rightsquigarrow^{\downarrow} v$ iff there exist $u'$, $v'$, $\rho$ and a definition $u' \equiv t$ in $D$ s.t. $u = u'[\rho]$, $v = v'[\rho]$ and $v' \in \mathsf{NbiTps}\,(t) \dot{\cup} \mathsf{NbiCinsts}\,(t)$

Faulty lemma would imply that that $\rightsquigarrow^{\Downarrow} \subseteq \rightsquigarrow^{\downarrow}$.

Fortunately, we can prove $\rightsquigarrow^{\Downarrow} \subseteq \rightsquigarrow^{\downarrow} \cup R$ where $R$ terminating and $\rightsquigarrow^{\downarrow} \circ R \subseteq \rightsquigarrow^{\downarrow} \cup R$.

Hence termination of $\rightsquigarrow^{\Downarrow}$ follows from that of $\rightsquigarrow^{\downarrow}$.

# Epilogue

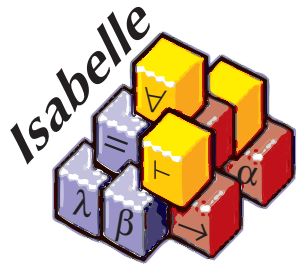



(work in progress)

$\Downarrow$

Proof of Consistency

$\Downarrow$




$\Downarrow$

# Some results about the variations of Higher-Order Logic underlying proof assistants

or

How I Learned to Start Worrying and Became Less Productive

Andrei Popescu
Middlesex University London

Joint work with Ondřej Kunčar