Admissible Types-to-PERs Relativization

in

Higher-Order Logic

Andrei Popescu

University of Sheffield

Dmitriy Traytel

University of Copenhagen

Proof Assistants, a.k.a. Interactive Theorem Provers

ACL2 Agda Coq HOL4 HOL Light HOL-ω

Isabelle/HOL Lean Mizar Nuprl Matita PVS

Proof Assistants, a.k.a. Interactive Theorem Provers

ACL2 Agda Coq HOL4 HOL Light HOL-ω

Isabelle/HOL Lean Mizar Nuprl Matita PVS

Logical foundations: a form of Higher-Order Logic (HOL) or Dependent Type Theory

Proof Assistants, a.k.a. Interactive Theorem Provers

ACL2 · Agda · Coq · HOL4 · HOL Light · HOL-ω

Isabelle/HOL · Lean · Mizar · Nuprl · Matita · PVS

Logical foundations: a form of Higher-Order Logic (HOL) or Dependent Type Theory

Verification Success Stories

Making the news: seL4, CompCert, CakeML,
Kepler's, Four Color, Odd Order, Gödel's theorems

Verification Success Stories

Making the news:

seL4, CompCert, CakeML,
Kepler's, Four Color, Odd Order, Gödel's theorems

Verification Success Stories

Making the news:

seL4, CompCert, CakeML,
Kepler's, Four Color, Odd Order, Gödel's theorems

Infrastructure:

Libraries of mathematical results
Automated proof methods
Abstraction/modularization mechanisms
Smart prover IDE

Verification Success Stories

Making the news:

seL4, CompCert, CakeML,
Kepler’s, Four Color, Odd Order, Gödel’s theorems

Infrastructure:

Libraries of mathematical results
Automated proof methods
Abstraction/modularization mechanisms
Smart prover IDE

Two Little Pigs

Two Little Pigs

group : $(\alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow \text{bool}$

Two Little Pigs

$\text{group} : (\alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow \text{bool}$

$$
\begin{aligned}
&\text{group}\ (*)\ e = \\
&(\forall x_\alpha, y_\alpha, z_\alpha.\ (x * y) * z = x * (y * z)) \wedge \\
&(\forall x_\alpha.\ x * e = x \wedge e * x = x) \wedge \\
&(\forall x_\alpha.\ \exists y_\alpha.\ x * y = e \wedge y * x = e)
\end{aligned}
$$

Two Little Pigs

group : $(\alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow \text{bool}$

group$^{\text{rlt}}$: $\alpha\ \text{set} \Rightarrow (\alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow \text{bool}$

group $(\,*\,)\ e =$
$(\forall x_\alpha, y_\alpha, z_\alpha.\ (x * y) * z = x * (y * z)) \wedge$
$(\forall x_\alpha.\ x * e = x \wedge e * x = x) \wedge$
$(\forall x_\alpha.\ \exists y_\alpha.\ x * y = e \wedge y * x = e)$

Two Little Pigs

$\text{group} : (\alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow \text{bool}$

$\text{group}\ (*)\ e =$
$(\forall x_\alpha, y_\alpha, z_\alpha.\ (x * y) * z = x * (y * z)) \wedge$
$(\forall x_\alpha.\ x * e = x \wedge e * x = x) \wedge$
$(\forall x_\alpha.\ \exists y_\alpha.\ x * y = e \wedge y * x = e)$

$\text{group}^{\text{rlt}} : \alpha\ \text{set} \Rightarrow (\alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow \text{bool}$

$\text{group}^{\text{rlt}}\ A\ (*)\ e =$
$(\forall x, y, z \in A.\ (x * y) * z = x * (y * z)) \wedge$
$(\forall x \in A.\ x * e = x \wedge e * x = x) \wedge$
$(\forall x \in A.\ \exists y \in A.\ x * y = e \wedge y * x = e)$

Two Little Pigs

group : $(\alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow$ bool

group $(*)\ e =$
$(\forall x_\alpha, y_\alpha, z_\alpha.\ (x * y) * z = x * (y * z)) \wedge$
$(\forall x_\alpha.\ x * e = x \wedge e * x = x) \wedge$
$(\forall x_\alpha.\ \exists y_\alpha.\ x * y = e \wedge y * x = e)$

proved Abel-Galois-Klein-Lie-Ruffini
in 10 kLoI and 2 person months

group$^{\mathrm{rlt}}$: α set $\Rightarrow (\alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow$ bool

group$^{\mathrm{rlt}}$ $A\ (*)\ e =$
$(\forall x, y, z \in A.\ (x * y) * z = x * (y * z)) \wedge$
$(\forall x \in A.\ x * e = x \wedge e * x = x) \wedge$
$(\forall x \in A.\ \exists y \in A.\ x * y = e \wedge y * x = e)$

Two Little Pigs

$\text{group} : (\alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow \text{bool}$

$\text{group}\ (*)\ e =$
$(\forall x_\alpha, y_\alpha, z_\alpha.\ (x * y) * z = x * (y * z)) \wedge$
$(\forall x_\alpha.\ x * e = x \wedge e * x = x) \wedge$
$(\forall x_\alpha.\ \exists y_\alpha.\ x * y = e \wedge y * x = e)$

proved Abel-Galois-Klein-Lie-Ruffini
in 10 kLoI and 2 person months

to use AGKLR for symmetric groups:

1. $\alpha\ \text{perm} = \{f :: \alpha \Rightarrow \alpha.\ \text{bij}\ f\}$
2. lift results about bijections to α perm
   (10 kLoI and 2 person months)

$\text{group}^{\text{rlt}} : \alpha\ \text{set} \Rightarrow (\alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow \text{bool}$

$\text{group}^{\text{rlt}}\ A\ (*)\ e =$
$(\forall x, y, z \in A.\ (x * y) * z = x * (y * z)) \wedge$
$(\forall x \in A.\ x * e = x \wedge e * x = x) \wedge$
$(\forall x \in A.\ \exists y \in A.\ x * y = e \wedge y * x = e)$

Two Little Pigs

group : $(\alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow \text{bool}$

group $(*)\ e =$
$(\forall x_\alpha, y_\alpha, z_\alpha.\ (x * y) * z = x * (y * z)) \wedge$
$(\forall x_\alpha.\ x * e = x \ \wedge\ e * x = x) \wedge$
$(\forall x_\alpha.\ \exists y_\alpha.\ x * y = e \ \wedge\ y * x = e)$

proved Abel-Galois-Klein-Lie-Ruffini
in 10 kLoI and 2 person months

to use AGKLR for symmetric groups:

1. α perm $= \{f :: \alpha \Rightarrow \alpha.\ \text{bij}\ f\}$
2. lift results about bijections to α perm
   (10 kLoI and 2 person months)

$\text{group}^{\text{rlt}}$: α set $\Rightarrow (\alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow \text{bool}$

$\text{group}^{\text{rlt}}\ A\ (*)\ e =$
$(\forall x, y, z \in A.\ (x * y) * z = x * (y * z)) \wedge$
$(\forall x \in A.\ x * e = x \ \wedge\ e * x = x) \wedge$
$(\forall x \in A.\ \exists y \in A.\ x * y = e \ \wedge\ y * x = e)$

proved Abel-Galois-Klein-Lie-Ruffini
assuming $e \in A$ and $\forall x, y \in A.\ x * y \in A$
in 30 kLoI and 6 person months

Two Little Pigs

group : $(\alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow$ bool

group $(*)$ $e =$
$(\forall x_\alpha, y_\alpha, z_\alpha.\ (x * y) * z = x * (y * z)) \wedge$
$(\forall x_\alpha.\ x * e = x \wedge e * x = x) \wedge$
$(\forall x_\alpha.\ \exists y_\alpha.\ x * y = e \wedge y * x = e)$

proved Abel-Galois-Klein-Lie-Ruffini
in 10 kLoI and 2 person months

to use AGKLR for symmetric groups:

1. α perm $= \{f :: \alpha \Rightarrow \alpha.$ bij $f\}$
2. lift results about bijections to α perm (10 kLoI and 2 person months)

group$^{\text{rlt}}$: α set $\Rightarrow (\alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow$ bool

group$^{\text{rlt}}$ A $(*)$ $e =$
$(\forall x, y, z \in A.\ (x * y) * z = x * (y * z)) \wedge$
$(\forall x \in A.\ x * e = x \wedge e * x = x) \wedge$
$(\forall x \in A.\ \exists y \in A.\ x * y = e \wedge y * x = e)$

proved Abel-Galois-Klein-Lie-Ruffini
assuming $e \in A$ and $\forall x, y \in A.\ x * y \in A$
in 30 kLoI and 6 person months

to use AGKLR for symmetric groups:

1. $A = \{f :: \alpha \Rightarrow \alpha.$ bij $f\}$
2. nothing left to do

Two Little Pigs

Type-based vs. Set-based

group : $(\alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow$ bool

group $(*)\ e =$
$(\forall x_\alpha, y_\alpha, z_\alpha.\ (x * y) * z = x * (y * z)) \wedge$
$(\forall x_\alpha.\ x * e = x \wedge e * x = x) \wedge$
$(\forall x_\alpha.\ \exists y_\alpha.\ x * y = e \wedge y * x = e)$

proved Abel-Galois-Klein-Lie-Ruffini
in 10 kLoI and 2 person months

to use AGKLR for symmetric groups:

1. α perm $= \{f :: \alpha \Rightarrow \alpha.\ \text{bij}\ f\}$
2. lift results about bijections to α perm
(10 kLoI and 2 person months)

group$^{\text{rlt}}$: α set $\Rightarrow (\alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow$ bool

group$^{\text{rlt}}$ $A\ (*)\ e =$
$(\forall x, y, z \in A.\ (x * y) * z = x * (y * z)) \wedge$
$(\forall x \in A.\ x * e = x \wedge e * x = x) \wedge$
$(\forall x \in A.\ \exists y \in A.\ x * y = e \wedge y * x = e)$

proved Abel-Galois-Klein-Lie-Ruffini
assuming $e \in A$ and $\forall x, y \in A.\ x * y \in A$
in 30 kLoI and 6 person months

to use AGKLR for symmetric groups:

1. $A = \{f :: \alpha \Rightarrow \alpha.\ \text{bij}\ f\}$
2. nothing left to do

Our Contribution

A logically safe mechanism for reducing the bureaucracy of formal developments in HOL-based proof assistants

HOL in One Line

Rank-1 Polymorphic Simply-Typed λ-Calculus with Hilbert Choice and Infinity

HOL in One Slide

Types

$\tau = \alpha \mid (\tau, \ldots, \tau)\ \kappa$ *bool* *ind* $\Rightarrow$

HOL in One Slide

Types

$\tau = \alpha \mid (\tau, \ldots, \tau)\ \kappa$ *bool* *ind* $\Rightarrow$

Terms

$t = x_\tau \mid c_\tau \mid t\ t \mid \lambda x_\tau.\ t$ $=_{\alpha \Rightarrow \alpha \Rightarrow bool}$ $\text{choice}_{(\alpha \Rightarrow bool) \Rightarrow \alpha}$

HOL in One Slide

Types

$\tau = \alpha \mid (\tau, \ldots, \tau)\ \kappa$ $\quad$ *bool* $\quad$ *ind* $\quad$ $\Rightarrow$

Terms

$t = x_\tau \mid c_\tau \mid t\ t \mid \lambda x_\tau.\ t$ $\quad$ $=_{\alpha\Rightarrow\alpha\Rightarrow bool}$ $\quad$ $\text{choice}_{(\alpha\Rightarrow bool)\Rightarrow\alpha}$

Definitions

Constant $c_\tau \equiv t$

Type $t_{\sigma\Rightarrow bool}\ x \longrightarrow \tau \equiv^{\text{Rep}_{\tau\Rightarrow\sigma}}_{\text{Abs}_{\sigma\Rightarrow\tau}}\ t$

$\text{True}_{bool} \equiv (\lambda x_{bool}.\ x) = (\lambda x.\ x)$

$\text{All}_{(\alpha\Rightarrow bool)\Rightarrow bool} \equiv \lambda p.\ (p = (\lambda x_\alpha.\ \text{True}))$

$\text{False}_{bool} \equiv \text{All}\ (\lambda x_{bool}.\ x)$

HOL in One Slide

Types

$\tau = \alpha \mid (\tau, \ldots, \tau)\ \kappa$ *bool*   *ind* $\Rightarrow$

Terms

$t = x_\tau \mid c_\tau \mid t\ t \mid \lambda x_\tau.\ t$ $=_{\alpha \Rightarrow \alpha \Rightarrow bool}$ $\text{choice}_{(\alpha \Rightarrow bool) \Rightarrow \alpha}$

Definitions

Constant $c_\tau \equiv t$

Type $t_{\sigma \Rightarrow bool}\ x \longrightarrow \tau \equiv^{\text{Rep}_{\tau \Rightarrow \sigma}}_{\text{Abs}_{\sigma \Rightarrow \tau}} t$

$\text{True}_{bool} \equiv (\lambda x_{bool}.\ x) = (\lambda x.\ x)$

$\text{All}_{(\alpha \Rightarrow bool) \Rightarrow bool} \equiv \lambda p.\ (p = (\lambda x_\alpha.\ \text{True}))$

$\text{False}_{bool} \equiv \text{All}\ (\lambda x_{bool}.\ x)$

Axioms

refl $x_\alpha = x$

sub $x_\alpha = y \longrightarrow p\ x \longrightarrow p\ y$

inf $\exists z, s.\ (\forall x.\ \neg\ (s\ x = z)) \wedge (\forall x, y.\ s\ x = s\ y \longrightarrow x = y)$

ch $p_{\alpha \Rightarrow bool}\ x \longrightarrow p\ (\text{choice}\ p)$

HOL in One Slide

Types

$\tau = \alpha \mid (\tau, \ldots, \tau)\, \kappa$ $\quad$ *bool* $\quad$ *ind* $\quad \Rightarrow$

Terms

$t = x_\tau \mid c_\tau \mid t\, t \mid \lambda x_\tau.\, t$ $\quad =_{\alpha \Rightarrow \alpha \Rightarrow bool}$ $\quad \mathsf{choice}_{(\alpha \Rightarrow bool) \Rightarrow \alpha}$

Definitions

Constant $c_\tau \equiv t$

Type $t_{\sigma \Rightarrow bool}\; x \longrightarrow \tau \equiv^{\mathsf{Rep}_{\tau \Rightarrow \sigma}}_{\mathsf{Abs}_{\sigma \Rightarrow \tau}} t$

$\mathsf{True}_{bool} \equiv (\lambda x_{bool}.\, x) = (\lambda x.\, x)$

$\mathsf{All}_{(\alpha \Rightarrow bool) \Rightarrow bool} \equiv \lambda p.\, (p = (\lambda x_\alpha.\, \mathsf{True}))$

$\mathsf{False}_{bool} \equiv \mathsf{All}\ (\lambda x_{bool}.\, x)$

Axioms

refl $\quad x_\alpha = x$

sub $\quad x_\alpha = y \longrightarrow p\, x \longrightarrow p\, y$

inf $\quad \exists z, s.\ (\forall x.\ \neg\ (s\, x = z)) \wedge (\forall x, y.\ s\, x = s\, y \longrightarrow x = y)$

ch $\quad p_{\alpha \Rightarrow bool}\; x \longrightarrow p\ (\mathsf{choice}\ p)$

Deduction

$$\frac{\varphi \in D \cup \Gamma}{D; \Gamma \vdash \varphi} \qquad \frac{D; \Gamma \vdash \varphi \qquad \alpha \notin \Gamma}{D; \Gamma \vdash \varphi[\sigma/\alpha]} \qquad \frac{D; \Gamma \vdash \varphi \qquad x_\sigma \notin \Gamma}{D; \Gamma \vdash \varphi[t/x_\sigma]} \qquad \frac{}{D; \Gamma \vdash (\lambda x.\ t)\ s = t[s/x_\sigma]}$$

$$\frac{D; \Gamma \vdash \varphi \longrightarrow \psi \qquad D; \Gamma \vdash \varphi}{D; \Gamma \vdash \psi} \qquad \frac{D; \Gamma \cup \{\varphi\} \vdash \psi}{D; \Gamma \vdash \varphi \longrightarrow \psi} \qquad \frac{D; \Gamma \vdash f\, x_\sigma = g\, x_\sigma \qquad x_\sigma \notin \Gamma}{D; \Gamma \vdash f = g}$$

HOL in Practice

And that's it! No induction, no datatypes, no recursion in the kernel!

These high-level mechanisms are defined from the (non-recursive) HOL primitives.

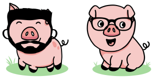
↓

Fewer opportunities to have soundness bugs in the kernel.

Back to  

Type-based formulation φ and its relativized set-based counterpart φ^{rlt}:

φ	$\forall e_\alpha.\ \forall *_{\alpha\Rightarrow\alpha\Rightarrow\alpha}.\ \text{group}\ (*)\ e \longrightarrow (\forall x_\alpha, y_\alpha, z_\alpha.\ x * y = x * z \longrightarrow y = z)$
φ^{rlt}	$\forall e_\alpha.\ *_{\alpha\Rightarrow\alpha\Rightarrow\alpha}.\ \forall A_{\alpha\ \text{set}}.\ \text{closed}\ A\ (*)\ e \longrightarrow$ $\text{group}^{\text{rlt}}\ A\ (*)\ e \longrightarrow (\forall x_\alpha, y_\alpha, z_\alpha{\in}A.\ x * y = x * z \longrightarrow y = z)$

Back to 

Type-based formulation φ and its relativized set-based counterpart φ^{rlt}:

φ	$\forall e_\alpha.\ \forall *_{\alpha\Rightarrow\alpha\Rightarrow\alpha}.\ \mathsf{group}\ (*)\ e \longrightarrow (\forall x_\alpha, y_\alpha, z_\alpha.\ x * y = x * z \longrightarrow y = z)$
φ^{rlt}	$\forall e_\alpha.\ *_{\alpha\Rightarrow\alpha\Rightarrow\alpha}.\ \forall A_{\alpha\ \mathsf{set}}.\ \mathsf{closed}\ A\ (*)\ e \longrightarrow$ $\mathsf{group}^{\mathsf{rlt}}\ A\ (*)\ e \longrightarrow (\forall x_\alpha, y_\alpha, z_\alpha \in A.\ x * y = x * z \longrightarrow y = z)$

Our goal: Formally (and automatically) infer φ^{rlt} from φ.

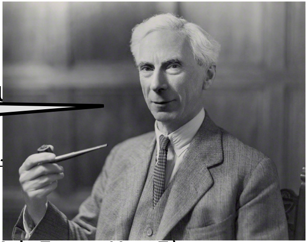
Back to  

Type-based formulation φ and its relativized set-based counterpart φ^{rlt}:

φ	$\forall e_\alpha.\ \forall *_{\alpha\Rightarrow\alpha\Rightarrow\alpha}.\ \text{group}\ (*)\ e \longrightarrow (\forall x_\alpha, y_\alpha, z_\alpha.\ x * y = x * z \longrightarrow y = z)$
φ^{rlt}	$\forall e_\alpha.\ *_{\alpha\Rightarrow\alpha\Rightarrow\alpha}.\ \forall A_{\alpha\ \text{set}}.\ \text{closed}\ A\ (*)\ e \longrightarrow$ $\text{group}^{\text{rlt}}\ A\ (*)\ e \longrightarrow (\forall x_\alpha, y_\alpha, z_\alpha \in A.\ x * y = x * z \longrightarrow y = z)$

Our goal: Formally (and automatically) infer φ^{rlt} from φ.
Is this doable without extending the HOL axiomatic base?

Back to  



Type- … sed cou …

> The method of "postulating" what we want has many advantages; they are the same as the advantages of theft over honest toil.

$z_\alpha .\ x\ *$ …

φ^{rlt}

$\forall e_\alpha .\ *_{\alpha \Rightarrow \alpha \Rightarrow \alpha} .\ \forall A_{\alpha\,\mathrm{set}} .\ \mathrm{closed}\ A\ (*)\ e \longrightarrow$
$\mathrm{group}^{\mathrm{rlt}}\ A\ (*)\ e \longrightarrow (\forall x_\alpha, y_\alpha, z_\alpha \in A .\ x * y = x * z \longrightarrow y = z)$

Our goal: Formally (and automatically) infer φ^{rlt} from φ.
Is this doable without extending the HOL axiomatic base?

Back to  

Type-based formulation φ and its relativized set-based counterpart φ^{rlt}:

φ	$\forall e_\alpha.\ \forall *_{\alpha\Rightarrow\alpha\Rightarrow\alpha}.\ \text{group}\ (*)\ e \longrightarrow (\forall x_\alpha, y_\alpha, z_\alpha.\ x * y = x * z \longrightarrow y = z)$
φ^{rlt}	$\forall e_\alpha.\ *_{\alpha\Rightarrow\alpha\Rightarrow\alpha}.\ \forall A_{\alpha\ \text{set}}.\ \text{closed}\ A\ (*)\ e \longrightarrow$ $\text{group}^{\text{rlt}}\ A\ (*)\ e \longrightarrow (\forall x_\alpha, y_\alpha, z_\alpha \in A.\ x * y = x * z \longrightarrow y = z)$

Our goal: Formally (and automatically) infer φ^{rlt} from φ.
Is this doable without extending the HOL axiomatic base?

But first: It was not clear how φ^{rlt} looks like in general.
What *structure* and *properties* do we need for relativization?

Towards a General Definition of Types-To-Sets

Example: the second-order property of left-inverse function uniqueness

$$\varphi \quad \begin{array}{|l} \forall e_\alpha.\ \forall *_{\alpha \Rightarrow \alpha \Rightarrow \alpha}.\ \text{group}\ (*)\ e \longrightarrow \\ (\forall inv_{\alpha \Rightarrow \alpha}, inv'_{\alpha \Rightarrow \alpha}.\ (\forall x_\alpha.\ inv\ x * x = e) \wedge (\forall x_\alpha.\ inv'\ x * x = e) \longrightarrow inv = inv') \\ \\ \end{array}$$

Towards a General Definition of Types-To-Sets

Example: the second-order property of left-inverse function uniqueness

φ	$\forall e_\alpha.\ \forall *_{\alpha\Rightarrow\alpha\Rightarrow\alpha}.\ \mathrm{group}\ (*)\ e \longrightarrow$ $(\forall inv_{\alpha\Rightarrow\alpha}, inv'_{\alpha\Rightarrow\alpha}.\ (\forall x_\alpha.\ inv\ x * x = e) \wedge (\forall x_\alpha.\ inv'\ x * x = e) \longrightarrow inv = inv')$
φ^{rlt}	$\forall e_\alpha.\ \forall A_{\alpha\ \mathrm{set}}.\ \mathrm{closed}\ A\ (*)\ e\ \wedge\ \mathrm{group}^{\mathrm{rlt}}\ A\ (*)\ e \longrightarrow$ $(\forall inv_{\alpha\Rightarrow\alpha}, inv'_{\alpha\Rightarrow\alpha} \in A \Rightarrow A.$ $(\forall x_\alpha \in A.\ inv\ x * x = e) \wedge (\forall x_\alpha \in A.\ inv'\ x * x = e) \longrightarrow inv =^{A} inv')$

Towards a General Definition of Types-To-Sets

Example: the second-order property of left-inverse function uniqueness

φ	$\forall e_\alpha.\ \forall *_{\alpha\Rightarrow\alpha\Rightarrow\alpha}.\ \text{group}\ (*)\ e \longrightarrow$ $(\forall inv_{\alpha\Rightarrow\alpha}, inv'_{\alpha\Rightarrow\alpha}.\ (\forall x_\alpha.\ inv\ x * x = e) \wedge (\forall x_\alpha.\ inv'\ x * x = e) \longrightarrow inv = inv')$
φ^{rlt}	$\forall e_\alpha.\ \forall A_{\alpha\ \text{set}}.\ \text{closed}\ A\ (*)\ e\ \wedge\ \text{group}^{\text{rlt}}\ A\ (*)\ e \longrightarrow$ $(\forall inv_{\alpha\Rightarrow\alpha}, inv'_{\alpha\Rightarrow\alpha} \in A \Rightarrow A.$ $(\forall x_\alpha \in A.\ inv\ x * x = e) \wedge (\forall x_\alpha \in A.\ inv'\ x * x = e) \longrightarrow inv =^{A} inv')$

Need set-based counterparts of the type constructors, incl. function space.

Towards a General Definition of Types-To-Sets

Example: the second-order property of left-inverse function uniqueness

φ	$\forall e_\alpha.\ \forall *_{\alpha\Rightarrow\alpha\Rightarrow\alpha}.\ \mathsf{group}\ (*)\ e \longrightarrow$ $(\forall inv_{\alpha\Rightarrow\alpha}, inv'_{\alpha\Rightarrow\alpha}.\ (\forall x_\alpha.\ inv\ x * x = e) \wedge (\forall x_\alpha.\ inv'\ x * x = e) \longrightarrow inv = inv')$
φ^{rlt}	$\forall e_\alpha.\ \forall A_{\alpha\ \mathsf{set}}.\ \mathsf{closed}\ A\ (*)\ e\ \wedge\ \mathsf{group}^{\mathsf{rlt}}\ A\ (*)\ e \longrightarrow$ $(\forall inv_{\alpha\Rightarrow\alpha}, inv'_{\alpha\Rightarrow\alpha} \in A \Rightarrow A.$ $(\forall x_\alpha \in A.\ inv\ x * x = e) \wedge (\forall x_\alpha \in A.\ inv'\ x * x = e) \longrightarrow inv =^{A} inv')$

Need set-based counterparts of the type constructors, incl. function space.

Need to relax the equality relation on higher-order types

Towards a General Definition of Types-To-Sets

Example: the second-order property of left-inverse function uniqueness

φ	$\forall e_\alpha.\ \forall *_{\alpha\Rightarrow\alpha\Rightarrow\alpha}.\ \mathsf{group}\ (*)\ e \longrightarrow$ $(\forall inv_{\alpha\Rightarrow\alpha}, inv'_{\alpha\Rightarrow\alpha}.\ (\forall x_\alpha.\ inv\ x * x = e) \wedge (\forall x_\alpha.\ inv'\ x * x = e) \longrightarrow inv = inv')$
φ^{rlt}	$\forall e_\alpha.\ \forall A_{\alpha\ \mathsf{set}}.\ \mathsf{closed}\ A\ (*)\ e\ \wedge\ \mathsf{group}^{\mathsf{rlt}}\ A\ (*)\ e \longrightarrow$ $(\forall inv_{\alpha\Rightarrow\alpha}, inv'_{\alpha\Rightarrow\alpha} \in A \Rightarrow A.$ $(\forall x_\alpha \in A.\ inv\ x * x = e) \wedge (\forall x_\alpha \in A.\ inv'\ x * x = e) \longrightarrow inv =^A inv')$

Need set-based counterparts of the type constructors, incl. function space.

Need to relax the equality relation on higher-order types
Relaxed equalities + restricted domains = PERs (partial equivalence relations)

Towards a General Definition of Types-To-Sets

Example: the second-order property of left-inverse function uniqueness

φ	$\forall e_\alpha.\ \forall *_{\alpha\Rightarrow\alpha\Rightarrow\alpha}.\ \text{group}\ (*)\ e \longrightarrow$ $(\forall inv_{\alpha\Rightarrow\alpha}, inv'_{\alpha\Rightarrow\alpha}.\ (\forall x_\alpha.\ inv\ x * x = e) \wedge (\forall x_\alpha.\ inv'\ x * x = e) \longrightarrow inv = inv')$
φ^{rlt}	$\forall e_\alpha.\ \forall R_{\alpha\ \text{rel}}.\ \text{per}\ R\ \wedge\ (*) \in \text{Dom}(R \Rightarrow R \Rightarrow R)\ \wedge\ e \in \text{Dom}(R)\ \wedge\ \text{group}^{\text{rlt}}\ R\ (*)\ e \longrightarrow$ $(\forall inv_{\alpha\Rightarrow\alpha}, inv'_{\alpha\Rightarrow\alpha} \in \text{Dom}(R \Rightarrow R).$ $(\forall x_\alpha \in \text{Dom}(R).\ inv\ x * x = e) \wedge (\forall x_\alpha \in \text{Dom}(R).\ inv'\ x * x = e) \longrightarrow (R \Rightarrow R)\ inv\ inv'$

Need set-based counterparts of the type constructors, incl. function space.

Need to relax the equality relation on higher-order types

Relaxed equalities + restricted domains = PERs (partial equivalence relations)

We generalize from Types-To-Sets to Types-To-PERs

Towards a General Definition of Types-To-Sets

Another example: iterated multiplication distributes over list append

$$\varphi \quad \begin{array}{l} \forall e_\alpha.\ \forall *_{\alpha \Rightarrow \alpha \Rightarrow \alpha}.\ \text{group}\ (*)\ e \longrightarrow \\ (\forall xs_{\alpha\ \text{list}}, ys_{\alpha\ \text{list}}.\ \text{fold}\ (*)\ e\ (\text{append}\ xs\ ys) = \text{fold}\ (*)\ e\ xs\ *\ \text{fold}\ (*)\ e\ ys) \end{array}$$

Towards a General Definition of Types-To-Sets

Another example: iterated multiplication distributes over list append

φ	$\forall e_\alpha.\ \forall *_{\alpha\Rightarrow\alpha\Rightarrow\alpha}.\ \mathsf{group}\ (*)\ e \longrightarrow$ $(\forall xs_{\alpha\ \mathsf{list}}, ys_{\alpha\ \mathsf{list}}.\ \mathsf{fold}\ (*)\ e\ (\mathsf{append}\ xs\ ys) = \mathsf{fold}\ (*)\ e\ xs\ *\ \mathsf{fold}\ (*)\ e\ ys)$
φ^{rlt}	$\forall e_\alpha.\ \forall R_{\alpha\ \mathsf{rel}}.\ \mathsf{per}\ R\ \wedge\ (*) \in \mathsf{Dom}(R \Rightarrow R \Rightarrow R)\ \wedge\ e \in \mathsf{Dom}(R)\ \wedge\ \mathsf{group}^{\mathsf{rlt}}\ R\ (*)\ e \longrightarrow$ $(\forall xs_{\alpha\ \mathsf{list}}, ys_{\alpha\ \mathsf{list}} \in \mathsf{Dom}(\mathsf{rel\_list}\ R).$ $\mathsf{fold}^{rlt}\ R\ (*)\ e\ (\mathsf{append}^{rlt}\ R\ xs\ ys) = \mathsf{fold}^{rlt}\ R\ (*)\ e\ xs\ *\ \mathsf{fold}^{rlt}\ R\ (*)\ e\ ys)$

Towards a General Definition of Types-To-Sets

Another example: iterated multiplication distributes over list append

φ	$\forall e_\alpha.\ \forall *_{\alpha\Rightarrow\alpha\Rightarrow\alpha}.\ \mathsf{group}\ (*)\ e \longrightarrow$ $(\forall xs_{\alpha\ \mathsf{list}}, ys_{\alpha\ \mathsf{list}}.\ \mathsf{fold}\ (*)\ e\ (\mathsf{append}\ xs\ ys) = \mathsf{fold}\ (*)\ e\ xs\ *\ \mathsf{fold}\ (*)\ e\ ys)$
φ^{rlt}	$\forall e_\alpha.\ \forall R_{\alpha\ \mathsf{rel}}.\ \mathsf{per}\ R\ \wedge\ (*) \in \mathsf{Dom}(R \Rightarrow R \Rightarrow R)\ \wedge\ e \in \mathsf{Dom}(R)\ \wedge\ \mathsf{group}^{\mathsf{rlt}}\ R\ (*)\ e \longrightarrow$ $(\forall xs_{\alpha\ \mathsf{list}}, ys_{\alpha\ \mathsf{list}} \in \mathsf{Dom}(\mathsf{rel\_list}\ R).$ $\mathsf{fold}^{rlt}\ R\ (*)\ e\ (\mathsf{append}^{rlt}\ R\ xs\ ys) = \mathsf{fold}^{rlt}\ R\ (*)\ e\ xs\ *\ \mathsf{fold}^{rlt}\ R\ (*)\ e\ ys)$

Want the relational interpretation of the container types to be the expected one (in spite of container types not being primitive in HOL)

Towards a General Definition of Types-To-Sets

Another example: iterated multiplication distributes over list append

φ	$\forall e_\alpha.\ \forall *_{\alpha\Rightarrow\alpha\Rightarrow\alpha}.\ \mathsf{group}\ (*)\ e \longrightarrow$ $(\forall xs_{\alpha\ \mathsf{list}}, ys_{\alpha\ \mathsf{list}}.\ \mathsf{fold}\ (*)\ e\ (\mathsf{append}\ xs\ ys) = \mathsf{fold}\ (*)\ e\ xs\ *\ \mathsf{fold}\ (*)\ e\ ys)$
φ^{rlt}	$\forall e_\alpha.\ \forall R_{\alpha\ \mathsf{rel}}.\ \mathsf{per}\ R\ \wedge\ (*) \in \mathsf{Dom}(R\Rightarrow R\Rightarrow R)\ \wedge\ e \in \mathsf{Dom}(R)\ \wedge\ \mathsf{group}^{\mathsf{rlt}}\ R\ (*)\ e \longrightarrow$ $(\forall xs_{\alpha\ \mathsf{list}}, ys_{\alpha\ \mathsf{list}} \in \mathsf{Dom}(\mathsf{rel\_list}\ R).$ $\mathsf{fold}^{rlt}\ R\ (*)\ e\ (\mathsf{append}^{rlt}\ R\ xs\ ys) = \mathsf{fold}^{rlt}\ R\ (*)\ e\ xs\ *\ \mathsf{fold}^{rlt}\ R\ (*)\ e\ ys)$

Want the relational interpretation of the container types to be the expected one (in spite of container types not being primitive in HOL)

Want that relationally parameteric functions are not affected by relativization.

Towards a General Definition of Types-To-Sets

Another example: iterated multiplication distributes over list append

φ	$\forall e_\alpha.\ \forall *_{\alpha\Rightarrow\alpha\Rightarrow\alpha}.\ \text{group}\ (*)\ e \longrightarrow$ $(\forall xs_{\alpha\ \text{list}}, ys_{\alpha\ \text{list}}.\ \text{fold}\ (*)\ e\ (\text{append}\ xs\ ys) = \text{fold}\ (*)\ e\ xs\ *\ \text{fold}\ (*)\ e\ ys)$
φ^{rlt}	$\forall e_\alpha.\ \forall R_{\alpha\ \text{rel}}.\ \text{per}\ R\ \wedge\ (*) \in \text{Dom}(R\Rightarrow R\Rightarrow R)\ \wedge\ e \in \text{Dom}(R)\ \wedge\ \text{group}^{\text{rlt}}\ R\ (*)\ e \longrightarrow$ $(\forall xs_{\alpha\ \text{list}}, ys_{\alpha\ \text{list}} \in \text{Dom}(\text{rel\_list}\ R).$ $\text{fold}\ (*)\ e\ (\text{append}\ xs\ ys) = \text{fold}\ (*)\ e\ xs\ *\ \text{fold}\ (*)\ e\ ys)$

Want the relational interpretation of the container types to be the expected one (in spite of container types not being primitive in HOL)

Want that relationally parameteric functions are not affected by relativization.

Towards a General Definition of Types-To-Sets

Another example: iterated multiplication distributes over list append

φ	$\forall e_\alpha.\ \forall *_{\alpha\Rightarrow\alpha\Rightarrow\alpha}.\ \text{group}\ (*)\ e \longrightarrow$ $(\forall xs_{\alpha\ \text{list}}, ys_{\alpha\ \text{list}}.\ \text{fold}\ (*)\ e\ (\text{append}\ xs\ ys) = \text{fold}\ (*)\ e\ xs * \text{fold}\ (*)\ e\ ys)$
φ^{rlt}	$\forall e_\alpha.\ \forall R_{\alpha\ \text{rel}}.\ \text{per}\ R \wedge (*) \in \text{Dom}(R\Rightarrow R\Rightarrow R) \wedge e \in \text{Dom}(R) \wedge \text{group}^{\text{rlt}}\ R\ (*)\ e \longrightarrow$ $(\forall xs_{\alpha\ \text{list}}, ys_{\alpha\ \text{list}} \in \text{Dom}(\text{rel\_list}\ R).$ $\text{fold}\ (*)\ e\ (\text{append}\ xs\ ys) = \text{fold}\ (*)\ e\ xs * \text{fold}\ (*)\ e\ ys)$

Want the relational interpretation of the container types to be the expected one (in spite of container types not being primitive in HOL)

Want that relationally parameteric functions are not affected by relativization.
Why? PER-relativization turns a term into its ”closest” PER-parametric counterpart.

Summary of the Desiderata for Relativization

Term relativization RLT and relational type interpretation RIN, such that:

1. RIN should be compatible with the container-type relators
2. RLT should generalize FOL-relativization
3. RLT(t) should recover t when instantiating relations to equalities
4. RLT(t) should be PER-parametric
5. Parametric terms t should not be affected by RLT
6. RLT should preserve provability (i.e., be admissible)

Defining RIN : Type ⟶ Term **and** RLT : Term ⟶ Term

$\mathsf{RIN}(\alpha) = R_{\alpha\ \mathsf{rel}}$ $\qquad \mathsf{RIN}(\mathsf{bool}) = =_{\mathsf{bool\ rel}}$ $\qquad \mathsf{RIN}(\sigma \Rightarrow \tau) = \mathsf{RIN}(\sigma) \Rightarrow \mathsf{RIN}(\tau)$

$$\mathsf{RIN}(\tau) = \mathsf{choice}\ (\lambda P_{\tau\ \mathsf{rel}}.\ \exists f_{\tau\Rightarrow\sigma}.\ \mathsf{bijUpto}\ f\ P\ \mathsf{RIN}(\sigma)_{\restriction \mathsf{RLT}(t)})$$

if $\tau \equiv t$ is a type definition where $t : \sigma \Rightarrow \mathsf{bool}$ and τ has the form $(\sigma_1, \ldots, \sigma_m)\kappa$

$\mathsf{RLT}(x_\sigma) = x_\sigma$ $\qquad \mathsf{RLT}(t_1\ t_2) = \mathsf{RLT}(t_1)\ \mathsf{RLT}(t_2)$ $\qquad \mathsf{RLT}(\lambda x_\sigma.\ t) = \lambda x_\sigma.\ \mathsf{RLT}(t)$

$$\mathsf{RLT}(=_{\sigma\Rightarrow\sigma\Rightarrow\mathsf{bool}}) = \mathsf{RIN}(\sigma)$$

$$\begin{aligned}\mathsf{RLT}(\mathsf{choice}_{(\sigma\Rightarrow\mathsf{bool})\Rightarrow\sigma}) = \lambda p_{\sigma\Rightarrow\mathsf{bool}}.\ &\mathsf{if}\ (\exists x_\sigma.\ \mathsf{RIN}(\sigma)\ x\ x \wedge p\ x)\\ &\mathsf{then\ choice}\ (\lambda x_\sigma.\ \mathsf{RIN}(\sigma)\ x\ x \wedge p\ x)\\ &\mathsf{else\ choice}\ (\lambda x_\sigma.\ \mathsf{RIN}(\sigma)\ x\ x \wedge \mathsf{RIN}(\sigma) \neq (=_{\sigma\ \mathsf{rel}}))\end{aligned}$$

Defining RIN : Type $\longrightarrow$ Term **and** RLT : Term $\longrightarrow$ Term

$$\text{RIN}(\alpha) = R_{\alpha\,\text{rel}} \qquad \text{RIN}(\text{bool}) \;=\; =_{\text{bool rel}} \qquad \text{RIN}(\sigma \Rightarrow \tau) \;=\; \text{RIN}(\sigma) \Rightarrow \text{RIN}(\tau)$$

$$\text{RIN}(\tau) \;=\; \text{choice}\;(\lambda P_{\tau\,\text{rel}}.\ \exists f_{\tau\Rightarrow\sigma}.\ \text{bijUpto}\ f\ P\ \text{RIN}(\sigma)_{\restriction \text{RLT}(t)})$$

if $\tau \equiv t$ is a type definition where $t : \sigma \Rightarrow \text{bool}$ and τ has the form $(\sigma_1, \ldots, \sigma_m)\kappa$

$$\text{RLT}(x_\sigma) = x_\sigma \qquad \text{RLT}(t_1\ t_2) = \text{RLT}(t_1)\ \text{RLT}(t_2) \qquad \text{RLT}(\lambda x_\sigma.\ t) = \lambda x_\sigma.\ \text{RLT}(t)$$

$$\text{RLT}(=_{\sigma\Rightarrow\sigma\Rightarrow\text{bool}}) = \text{RIN}(\sigma)$$

Equality mapped to the PER-relational interpretation

$$\text{RLT}(\text{choice}_{(\sigma\Rightarrow\text{bool})\Rightarrow\sigma}) = \lambda p_{\sigma\Rightarrow\text{bool}}.\ \text{if}\ (\exists x_\sigma.\ \text{RIN}(\sigma)\ x\ x \wedge p\ x)$$
$$\text{then choice}\ (\lambda x_\sigma.\ \text{RIN}(\sigma)\ x\ x \wedge p\ x)$$
$$\text{else choice}\ (\lambda x_\sigma.\ \text{RIN}(\sigma)\ x\ x \wedge \text{RIN}(\sigma) \neq (=_{\sigma\,\text{rel}}))$$

Defining RIN : Type → Term **and** RLT : Term → Term

$\mathsf{RIN}(\alpha) = R_{\alpha\,\mathsf{rel}}$ $\quad$ $\mathsf{RIN}(\mathsf{bool}) = {=_{\mathsf{bool\,rel}}}$ $\quad$ $\mathsf{RIN}(\sigma \Rightarrow \tau) = \mathsf{RIN}(\sigma) \Rightarrow \mathsf{RIN}(\tau)$

$\mathsf{RIN}(\tau) = \mathsf{choice}\ (\lambda P_{\tau\,\mathsf{rel}}.\ \exists f_{\tau\Rightarrow\sigma}.\ \mathsf{bijUpto}\ f\ P\ \mathsf{RIN}(\sigma)_{\restriction \mathsf{RLT}(t)})$
if $\tau \equiv t$ is a type definition where $t : \sigma \Rightarrow \mathsf{bool}$ and τ has the form $(\sigma_1, \ldots, \sigma_m)\kappa$

$\mathsf{RLT}(x_\sigma) = x_\sigma$ $\quad$ $\mathsf{RLT}(t_1\ t_2) = \mathsf{RLT}(t_1)\ \mathsf{RLT}(t_2)$ $\quad$ $\mathsf{RLT}(\lambda x_\sigma.\ t) = \lambda x_\sigma.\ \mathsf{RLT}(t)$

$\mathsf{RLT}(=_{\sigma\Rightarrow\sigma\Rightarrow\mathsf{bool}}) = \mathsf{RIN}(\sigma)$

$$\begin{aligned}\mathsf{RLT}(\mathsf{choice}_{(\sigma\Rightarrow\mathsf{bool})\Rightarrow\sigma}) = \lambda p_{\sigma\Rightarrow\mathsf{bool}}.\ &\mathsf{if}\ (\exists x_\sigma.\ \mathsf{RIN}(\sigma)\ x\ x \wedge p\ x)\\ &\mathsf{then\ choice}\ (\lambda x_\sigma.\ \mathsf{RIN}(\sigma)\ x\ x \wedge p\ x)\\ &\mathsf{else\ choice}\ (\lambda x_\sigma.\ \mathsf{RIN}(\sigma)\ x\ x \wedge \mathsf{RIN}(\sigma) \neq (=_{\sigma\,\mathsf{rel}}))\end{aligned}$$

The (notoriously non-parametric) Hilbert Choice needs special treatment

Defining RIN : Type ⟶ Term **and** RLT : Term ⟶ Term

$\mathsf{RIN}(\alpha) = R_{\alpha\ \mathsf{rel}}$ $\qquad \mathsf{RIN}(\mathsf{bool}) = =_{\mathsf{bool\ rel}}$ $\qquad \mathsf{RIN}(\sigma \Rightarrow \tau) = \mathsf{RIN}(\sigma) \Rightarrow \mathsf{RIN}(\tau)$

$$\mathsf{RIN}(\tau) = \mathsf{choice}\ (\lambda P_{\tau\ \mathsf{rel}}.\ \exists f_{\tau\Rightarrow\sigma}.\ \mathsf{bijUpto}\ f\ P\ \mathsf{RIN}(\sigma)_{\restriction \mathsf{RLT}(t)})$$

if $\tau \equiv t$ is a type definition where $t : \sigma \Rightarrow \mathsf{bool}$ and τ has the form $(\sigma_1, \ldots, \sigma_m)\kappa$

– Only behaves well when defining predicates t are "wide" enough: $|t| \geq |\mathsf{RLT}(t)/\mathsf{RIN}(\sigma)|$.

– Choice of P needs flexibility to correctly capture container types.

$\mathsf{RLT}(x_\sigma) = x_\sigma$ $\qquad \mathsf{RLT}(t_1\ t_2) = \mathsf{RLT}(t_1)\ \mathsf{RLT}(t_2)$ $\qquad \mathsf{RLT}(\lambda x_\sigma.\ t) = \lambda x_\sigma.\ \mathsf{RLT}(t)$

$$\mathsf{RLT}(=_{\sigma\Rightarrow\sigma\Rightarrow\mathsf{bool}}) = \mathsf{RIN}(\sigma)$$

$$\begin{aligned}\mathsf{RLT}(\mathsf{choice}_{(\sigma\Rightarrow\mathsf{bool})\Rightarrow\sigma}) = \lambda p_{\sigma\Rightarrow\mathsf{bool}}.\ &\mathsf{if}\ (\exists x_\sigma.\ \mathsf{RIN}(\sigma)\ x\ x \wedge p\ x) \\ &\mathsf{then\ choice}\ (\lambda x_\sigma.\ \mathsf{RIN}(\sigma)\ x\ x \wedge p\ x) \\ &\mathsf{else\ choice}\ (\lambda x_\sigma.\ \mathsf{RIN}(\sigma)\ x\ x \wedge \mathsf{RIN}(\sigma) \neq (=_{\sigma\ \mathsf{rel}}))\end{aligned}$$

Main Result

Desiderata:

1. RIN should be compatible with the container-type relators
2. RLT should generalize FOL-relativization
3. RLT(t) should recover t when instantiating relations to equalities
4. RLT(t) should be PER-parametric
5. Parametric terms t should not be affected by RLT
6. RLT should preserve provability (i.e., be admissible)

Main Result

Desiderata:

1. RIN should be compatible with the container-type relators
2. RLT should generalize FOL-relativization
3. RLT(t) should recover t when instantiating relations to equalities
4. RLT(t) should be PER-parametric
5. Parametric terms t should not be affected by RLT
6. RLT should preserve provability (i.e., be admissible)

A type definition $\tau \equiv t$ where $t : \sigma \Rightarrow \text{bool}$ is said to be *wide* if $|t| \geq |\mathsf{RLT}(t)/\mathsf{RIN}(\sigma)|$.

Main Result

Desiderata:

1. RIN should be compatible with the container-type relators
2. RLT should generalize FOL-relativization
3. RLT(t) should recover t when instantiating relations to equalities
4. RLT(t) should be PER-parametric
5. Parametric terms t should not be affected by RLT
6. RLT should preserve provability (i.e., be admissible)

A type definition $\tau \equiv t$ where $t : \sigma \Rightarrow$ bool is said to be *wide* if $|t| \geq |\mathrm{RLT}(t)/\mathrm{RIN}(\sigma)|$.

THEOREM: If the type-definitions in the background definitional theory are wide, then RLT and RIN satisfy the above desiderata.

Main Result

Desiderata:

1. RIN should be compatible with the container-type relators
2. RLT should generalize FOL-relativization
3. RLT(t) should recover t when instantiating relations to equalities
4. RLT(t) should be PER-parametric
5. Parametric terms t should not be affected by RLT
6. RLT should preserve provability (i.e., be admissible)

A type definition $\tau \equiv t$ where $t : \sigma \Rightarrow \text{bool}$ is said to be *wide* if $|t| \geq |\text{RLT}(t)/\text{RIN}(\sigma)|$.

THEOREM: If the type-definitions in the background definitional theory are wide, then RLT and RIN satisfy the above desiderata.

COROLLARY: (Types-To-Sets) is admissible assuming wide type definitions.

Main Result

Desiderata:

1. RIN should be compatible with the container-type relators
2. RLT should generalize FOL-relativization
3. RLT(t) should recover t when instantiating relations to equalities
4. RLT(t) should be PER-parametric
5. Parametric terms t should not be affected by RLT
6. RLT should preserve provability (i.e., be admissible)

A type definition $\tau \equiv t$ where $t : \sigma \Rightarrow \text{bool}$ is said to be *wide* if $|t| \geq |\text{RLT}(t)/\text{RIN}(\sigma)|$.

THEOREM: If the type-definitions in the background definitional theory are wide, then RLT and RIN satisfy the above desiderata.

COROLLARY: (Types-To-Sets) is admissible assuming wide type definitions.

Main Result

Desiderata:

1. RIN should be compatible with the container-type relators
2. RLT should generalize FOL-relativization
3. RLT(t) should recover t when instantiating relations to equalities
4. RLT(t) should be PER-parametric
5. Parametric terms t should not be affected by RLT
6. RLT should preserve provability (i.e., be admissible)

A type definition $\tau \equiv t$ where $t : \sigma \Rightarrow$ bool is said to be *wide* if $|t| \geq |\mathrm{RLT}(t)/\mathrm{RIN}(\sigma)|$.

THEOREM: If the type-definitions in the background definitional theory are wide, then RLT and RIN satisfy the above desiderata.

COROLLARY: (Types-To-Sets) is admissible assuming wide type definitions.

The Result's Scope

Empirical Conjecture: Type definitions of interest in HOL developments are wide.

The Result's Scope

Empirical Conjecture: Type definitions of interest in HOL developments are wide.

Evidence:

- type definitions involved in defining (co)datatypes (which are already 99% of the cases of interest) are wide

The Result's Scope

Empirical Conjecture: Type definitions of interest in HOL developments are wide.

Evidence:

- type definitions involved in defining (co)datatypes (which are already 99% of the cases of interest) are wide
- and so are the non-(co)datatypes from the Isabelle/HOL distribution

Relativization Infrastructure

```
definition group where
  "group tms e ⟷
     (∀x y z. tms (tms x y) z = tms x (tms y z)) ∧
     (∀x. tms x e = x ∧ tms e x = x) ∧
     (∀x. ∃y. tms x y = e ∧ tms y x = e)"

lemma lemma3_aux1:"group tms e ⟶ foldl tms x ys = tms x (foldl tms e ys)" [5 lines]
lemma lemma3_aux2: "group tms e ⟶ foldl tms x (xs @ ys) = tms (foldl tms x xs) (foldl tms e ys)" [1 lines]
lemma lemma3: "group tms e ⟶ (∀xs ys. foldl tms e (xs @ ys) = tms (foldl tms e xs) (foldl tms e ys))" [1 lines]

local_setup ‹RLCST @{term group}›

lemma group_rlt_alt:
  "neper R ⟹ group_rlt R tms e ⟷
    (∀x y z. R x x ∧ R y y ∧ R z z ⟶ R (tms (tms x y) z) (tms x (tms y z))) ∧
    (∀x. R x x ⟶ R (tms x e) x ∧ R (tms e x) x) ∧
    (∀x. R x x ⟶ (∃y. R y y ∧ R (tms x y) e ∧ R (tms y x) e))"
  unfolding group_rlt_def by auto

local_setup ‹RLTHM @{binding lemma3_rlt} @{thm lemma3}›

lemma lemma3_rlt_readable:
  assumes "neper R" "R e e" "(R ===> R ===> R) tms tms"
  shows "group_rlt R tms e ⟶
  (∀xs ys. rel_list R xs xs ⟶ rel_list R ys ys ⟶ R (foldl tms e (xs @ ys)) (tms (foldl tms e xs) (foldl tms e ys)))"
  supply assms(1)[simp] assms(1)[THEN list.rrel_neper, simp] using lemma3_rlt[OF assms, simplified]
  by (auto simp only: rrel_alt rlt_param neper assms(1))
```

Wideness Proofs

```
typedef (overloaded) 'a poly = "{f :: nat ⇒ 'a::zero. ∀∞ n. f n = 0}"
  morphisms coeff Abs_poly
  by (auto intro!: ALL_MOST)
```

200 lines later

```
wide_typedef poly rel: rel_poly rep: "λR1 R2. gg R1 R2 ∘ coeff"
  subgoal using rel_poly_neper .
  subgoal using rel_poly_eq .
  subgoal using bij_upto_transportFromRaw[OF poly.type_definition_poly rel_poly_def,
     unfolded isPoly_def[abs_def, symmetric], OF bij_upto_gg]
  unfolding isPoly_rlt_def .
  subgoal using gg_eq by simp .
```

Wideness Proofs

$\alpha\ \mathit{fset}^*$	finite sets over α
$\alpha\ \mathit{cset}^*$	countable sets over α
$\alpha\ \mathit{multiset}^*$	multisets (bags) over α
$(\alpha, \beta)\ \mathit{fmap}^*$	partial functions of finite support between α and β
$\alpha\ \mathit{biject}$	bijections on α
$\alpha\ \mathit{dlist}^*$	non-repetitive lists over α
$(\alpha, \beta)\ \mathit{alist}^*$	association lists with values in α and keys in β
$(\alpha, \beta)\ \mathit{node}$	the pre-datatype universe by Berghofer and Wenzel [1999]

$\alpha\ \mathit{filter}^*$	filters on the powerset of α
$\alpha\ \mathit{poly}$	polynomials with α-coefficients
$\alpha\ \mathit{fls}$	formal Laurent series with α-coefficients
$(\alpha, \beta)\ \mathit{comm}^*$	commutative α-operators with values in β
$\alpha\ \mathit{comparator}$	comparison functions on α
$\alpha\ \mathit{bit0},\ \alpha\ \mathit{bit1}$	finite types used for binary representation

,

Related Work

Relational interpretation

- Reynolds 1983. Types, abstraction, and parametric polymorphism
- Wadler 1989. Theorems for free!
- Bernardy et al. 2012. Extension to DTT

Previous work on Types-To-Sets in Isabelle

- Kunčar & Popescu 2015. Local Typedef axiom
- Immler & Zhan 2019. Divasón et al. 2020. Large case studies
- Milehins 2022. Types-To-Sets conversion tool

PER constructions in DTT

- Constable et al. 1986. Subset types and quotient types in Nuprl
- Barthe et al. 2003. (Partial) setoids in Agda and Coq
- Altenkirch et al. 2019. Types to setoids in Martin-Löf type theory

Take Home Message

Motto: Prove easily (type-based) and still be flexible (PER-based)!

Take Home Message

Motto: Prove easily (type-based) and still be flexible (PER-based)!

And don't worry:
Very likely (if only wide types are involved), you are not reasoning outside of HOL!

Admissible Types-to-PERs Relativization

in

Higher-Order Logic

Andrei Popescu

University of Sheffield

Dmitriy Traytel

University of Copenhagen