Morphisms between Dedukti theories

ICSPA Meeting 2026

Thomas Traversié
joint work with Florian Rabe (FAU Erlangen)

Interoperability between proof systems

- The **$\lambda\Pi$-calculus modulo rewriting** [Cousineau and Dowek, 2007]
  - LF ($= \lambda$-calculus $+$ dependent types) $+$ rewrite rules
  - Implemented in Dedukti [Saillard, 2015, Assaf et al, 2016]

Interoperability between proof systems

- The **$\lambda\Pi$-calculus modulo rewriting** [Cousineau and Dowek, 2007]
  – LF ($=$ λ-calculus $+$ dependent types) $+$ rewrite rules
  – Implemented in Dedukti [Saillard, 2015, Assaf et al, 2016]

- Dedukti $=$ **middleware** to exchange proofs to/from/inside Dedukti

Interoperability between proof systems

- The **$\lambda\Pi$-calculus modulo rewriting** [Cousineau and Dowek, 2007]
  – LF (= λ-calculus + dependent types) + rewrite rules
  – Implemented in Dedukti [Saillard, 2015, Assaf et al, 2016]

- Dedukti = **middleware** to exchange proofs to/from/inside Dedukti

- **Theory morphisms** = translation template between theories
  – Defined for LF [Harper et al, 1994]
  – Extended to Dedukti [Felicissimo, 2022]

Interoperability between proof systems

- The **$\lambda\Pi$-calculus modulo rewriting** [Cousineau and Dowek, 2007]
  - LF (= λ-calculus + dependent types) + rewrite rules
  - Implemented in Dedukti [Saillard, 2015, Assaf et al, 2016]

- Dedukti = **middleware** to exchange proofs to/from/inside Dedukti

- **Theory morphisms** = translation template between theories
  - Defined for LF [Harper et al, 1994]
  - Extended to Dedukti [Felicissimo, 2022]

Goal: apply theory morphisms to translate between Dedukti theories

The $\lambda\Pi$-calculus modulo rewriting

- Syntax

Objects	$M, N ::= c \mid x \mid \lambda x : A.\ M \mid M\ N$
Types	$A, B ::= a \mid \Pi x : A.\ B \mid \lambda x : A.\ B \mid A\ M$
Kinds	$K ::= \mathsf{Type} \mid \Pi x : A.\ K$
Terms	$t, u ::= M \mid A \mid K \mid \mathsf{Kind}$

Theories	$\mathbb{T} ::= \varnothing \mid \mathbb{T}, c : A \mid \mathbb{T}, a : K \mid \mathbb{T}, M \hookrightarrow N \mid \mathbb{T}, A \hookrightarrow B$

The $\lambda\Pi$-calculus modulo rewriting

- Syntax

$$
\begin{array}{lrl}
\textit{Objects} & M, N ::= & c \mid x \mid \lambda x : A.\ M \mid M\ N \\
\textit{Types} & A, B ::= & a \mid \Pi x : A.\ B \mid \lambda x : A.\ B \mid A\ M \\
\textit{Kinds} & K ::= & \mathsf{Type} \mid \Pi x : A.\ K \\
\textit{Terms} & t, u ::= & M \mid A \mid K \mid \mathsf{Kind}
\end{array}
$$

$$
\textit{Theories} \qquad \mathbb{T} ::= \varnothing \mid \mathbb{T}, c : A \mid \mathbb{T}, a : K \mid \mathbb{T}, M \hookrightarrow N \mid \mathbb{T}, A \hookrightarrow B
$$

- Terms considered modulo the **conversion** $\equiv_{\beta\mathcal{R}}$
  - Generated by β-reduction and rewrite rules
  - Reflexive, symmetric and transitive

Outline

Translation

- Principle: **represent** the theory $\mathbb{S}$ inside the theory $\mathbb{T}$

Translation

- Principle: **represent** the theory $\mathbb{S}$ inside the theory $\mathbb{T}$
- Translation μ

$$
\begin{array}{lll}
\mu(x) & = & x \\
\mu(c) & = & \mu_c \text{ (parameter)} \\
\mu(a) & = & \mu_a \text{ (parameter)} \\
\mu(M\ N) & = & \mu(M)\ \mu(N) \\
\mu(A\ M) & = & \mu(A)\ \mu(M) \\
\mu(\text{Type}) & = & \text{Type}
\end{array}
\qquad
\begin{array}{lll}
\mu(\lambda x : A.\ M) & = & \lambda x : \mu(A).\ \mu(M) \\
\mu(\lambda x : A.\ B) & = & \lambda x : \mu(A).\ \mu(B) \\
\mu(\Pi x : A.\ B) & = & \Pi x : \mu(A).\ \mu(B) \\
\mu(\Pi x : A.\ K) & = & \Pi x : \mu(A).\ \mu(K) \\
\mu(\text{Kind}) & = & \text{Kind}
\end{array}
$$

Definition

- **In LF**

μ is a **theory morphism** from $\mathbb{S}$ to $\mathbb{T}$ when

1. for every $c : A \in \mathbb{S}$, there exists a term μ_c such that $\vdash_{\mathbb{T}} \mu_c : \mu(A)$
2. for every $a : K \in \mathbb{S}$, there exists a term μ_a such that $\vdash_{\mathbb{T}} \mu_a : \mu(K)$

Definition

- **In LF**

μ is a **theory morphism** from $\mathbb{S}$ to $\mathbb{T}$ when

1. for every $c : A \in \mathbb{S}$, there exists a term μ_c such that $\vdash_{\mathbb{T}} \mu_c : \mu(A)$
2. for every $a : K \in \mathbb{S}$, there exists a term μ_a such that $\vdash_{\mathbb{T}} \mu_a : \mu(K)$

- **In the $\lambda\Pi$-calculus modulo rewriting**

Additionally, for every $\ell \hookrightarrow r \in \mathbb{S}$, we require in $\mathbb{T}$

✗ The rewrite rule $\mu(\ell) \hookrightarrow \mu(r)$

✗ The rewriting $\mu(\ell) \hookrightarrow^*_{\beta\mathcal{R}} \mu(r)$

✓ The conversion $\mu(\ell) \equiv_{\beta\mathcal{R}} \mu(r)$

Results

- **Convertibility is preserved** by theory morphisms
  1. If $A \equiv_{\beta\mathcal{R}} B$ in $\mathbb{S}$ then $\mu(A) \equiv_{\beta\mathcal{R}} \mu(B)$ in $\mathbb{T}$
  2. If $K \equiv_{\beta\mathcal{R}} K'$ in $\mathbb{S}$ then $\mu(K) \equiv_{\beta\mathcal{R}} \mu(K')$ in $\mathbb{T}$

Results

- **Convertibility is preserved** by theory morphisms
  1. If $A \equiv_{\beta\mathcal{R}} B$ in $\mathbb{S}$ then $\mu(A) \equiv_{\beta\mathcal{R}} \mu(B)$ in $\mathbb{T}$
  2. If $K \equiv_{\beta\mathcal{R}} K'$ in $\mathbb{S}$ then $\mu(K) \equiv_{\beta\mathcal{R}} \mu(K')$ in $\mathbb{T}$

- **Preservation theorem**
  1. If $\Gamma \vdash_{\mathbb{S}} M : A$ then $\mu(\Gamma) \vdash_{\mathbb{T}} \mu(M) : \mu(A)$
  2. If $\Gamma \vdash_{\mathbb{S}} A : K$ then $\mu(\Gamma) \vdash_{\mathbb{T}} \mu(A) : \mu(K)$
  3. If $\Gamma \vdash_{\mathbb{S}} K : \mathrm{Kind}$ then $\mu(\Gamma) \vdash_{\mathbb{T}} \mu(K) : \mathrm{Kind}$

Outline

Theory Morphisms

Case study: Sort-Erasure Translations

Implementation

Conclusion

Hard-Sorted logic `HFOL`

Terms are sorted with the typing relation of the framework (Church encoding)

Hard-Sorted logic HFOL

Terms are sorted with the typing relation of the framework (Church encoding)

Set : Type $\qquad$ El : $Set \to$ Type $\qquad$ $Prop$: Type $\qquad$ Prf : $Prop \to$ Type

Hard-Sorted logic HFOL

Terms are sorted with the typing relation of the framework (Church encoding)

$Set : \text{Type}$ $\quad El : Set \to \text{Type}$ $\quad Prop : \text{Type}$ $\quad Prf : Prop \to \text{Type}$

$\Rightarrow \, : Prop \to Prop \to Prop$

$Prf\ (p \Rightarrow q) \hookrightarrow Prf\ p \to Prf\ q$

Hard-Sorted logic HFOL

Terms are sorted with the typing relation of the framework (Church encoding)

Set : Type $\quad El : Set \to$ Type $\quad Prop$: Type $\quad Prf : Prop \to$ Type

$\Rightarrow : Prop \to Prop \to Prop$

$Prf\ (p \Rightarrow q) \hookrightarrow Prf\ p \to Prf\ q$

$\forall : \Pi a : Set.\ (El\ a \to Prop) \to Prop$

$\mathsf{all_i} : \Pi a : Set.\ \Pi p : El\ a \to Prop.\ (\Pi x : El\ a.\ Prf\ (p\ x)) \to Prf\ (\forall\ a\ p)$

$\mathsf{all_e} : \Pi a : Set.\ \Pi p : El\ a \to Prop.\ Prf\ (\forall\ a\ p) \to \Pi x : El\ a.\ Prf\ (p\ x)$

Soft-Sorted logic `SFOL`

Terms are sorted with an external relation (Curry encoding)

Soft-Sorted logic SFOL

Terms are sorted with an external relation (Curry encoding)

$$tm : \text{Type} \qquad Set : \text{Type} \qquad Prop : \text{Type} \qquad Prf : Prop \to \text{Type} \qquad \# : tm \to Set \to \text{Type}$$

Soft-Sorted logic SFOL

Terms are sorted with an external relation (Curry encoding)

$tm : \text{Type} \qquad Set : \text{Type} \qquad Prop : \text{Type} \qquad Prf : Prop \to \text{Type} \qquad \# : tm \to Set \to \text{Type}$

$\Rightarrow\ : Prop \to Prop \to Prop$

$Prf\ (p \Rightarrow q) \hookrightarrow Prf\ p \to Prf\ q$

Soft-Sorted logic SFOL

Terms are sorted with an external relation (Curry encoding)

$tm : \text{Type} \qquad Set : \text{Type} \qquad Prop : \text{Type} \qquad Prf : Prop \to \text{Type} \qquad \# : tm \to Set \to \text{Type}$

$\Rightarrow : Prop \to Prop \to Prop$

$Prf\ (p \Rightarrow q) \hookrightarrow Prf\ p \to Prf\ q$

$\forall : \Pi a : Set.\ (\Pi x : tm.\ x \# a \to Prop) \to Prop$

$Prf\ (\forall\ a\ p) \hookrightarrow \Pi x : tm.\ \Pi h : x \# a.\ Prf\ (p\ x\ h)$

From HFOL to SFOL

- **Intuition**: erasing the sorting information and recovering it later

From HFOL to SFOL

- **Intuition**: erasing the sorting information and recovering it later

  Problem: we need the sorting information for $\mu(\mathsf{all}_{\mathsf{e}})$

From HFOL to SFOL

- **Intuition**: erasing the sorting information and recovering it later

  Problem: we need the sorting information for $\mu(\mathsf{all_e})$

  Solution: we use **dependent pairs** to bundle together a term and its sorting information

From HFOL to SFOL

- **Intuition**: erasing the sorting information and recovering it later
  Problem: we need the sorting information for $\mu(\mathsf{all_e})$
  Solution: we use **dependent pairs** to bundle together a term and its sorting information

- Theory morphism:

  $\mu(El) = \lambda a : Set.\ \text{pair}\ a$

  $\mu(\forall) = \lambda a : Set.\ \lambda p : \text{pair}\ a \to Prop.\ \forall\ a\ (\lambda x.\ \lambda h.\ p\ (\text{mk\_pair}\ a\ x\ h))$

From HFOL to SFOL

- **Intuition**: erasing the sorting information and recovering it later
  Problem: we need the sorting information for $\mu(\mathsf{all_e})$
  Solution: we use **dependent pairs** to bundle together a term and its sorting information

- Theory morphism:

  $\mu(El) = \lambda a : Set.\ \mathsf{pair}\ a$

  $\mu(\forall) = \lambda a : Set.\ \lambda p : \mathsf{pair}\ a \to Prop.\ \forall\ a\ (\lambda x.\ \lambda h.\ p\ (\mathsf{mk\_pair}\ a\ x\ h))$

- $\mu(\Rightarrow) = \Rightarrow$ and its rewrite rule is trivially preserved

From HFOL to SFOL

- **Intuition**: erasing the sorting information and recovering it later
  Problem: we need the sorting information for $\mu(\mathsf{all}_{\mathsf{e}})$
  Solution: we use **dependent pairs** to bundle together a term and its sorting information

- Theory morphism:

$$\mu(El) = \lambda a : Set.\ \mathsf{pair}\ a$$

$$\mu(\forall) = \lambda a : Set.\ \lambda p : \mathsf{pair}\ a \to Prop.\ \forall\ a\ (\lambda x.\ \lambda h.\ p\ (\mathsf{mk\_pair}\ a\ x\ h))$$

- $\mu(\Rightarrow) = \Rightarrow$ and its rewrite rule is trivially preserved

- **Remark**: we cannot define the morphism if $\forall$ is defined with a rewrite rule in HFOL

Unsorted logic `UFOL`

All terms have the generic sort tm

Unsorted logic `UFOL`

All terms have the generic sort tm

$tm : \text{Type}$ $\qquad$ $Prop : \text{Type}$ $\qquad$ $Prf : Prop \to \text{Type}$

Unsorted logic UFOL

All terms have the generic sort tm

$tm : \text{Type}$ $\quad$ $Prop : \text{Type}$ $\quad$ $Prf : Prop \to \text{Type}$

$\Rightarrow : Prop \to Prop \to Prop$

$Prf\ (p \Rightarrow q) \hookrightarrow Prf\ p \to Prf\ q$

Unsorted logic UFOL

All terms have the generic sort tm

$tm : \text{Type}$ $\quad Prop : \text{Type}$ $\quad Prf : Prop \to \text{Type}$

$\Rightarrow : Prop \to Prop \to Prop$

$Prf\ (p \Rightarrow q) \hookrightarrow Prf\ p \to Prf\ q$

$\forall : (tm \to Prop) \to Prop$

$Prf\ (\forall\ p) \hookrightarrow \Pi x : tm.\ Prf\ (p\ x)$

From SFOL to UFOL

- **Intuition**: transforming sorts into sort predicates

$\mu(Set) = tm \to Prop$ $\mu(\#) = \lambda x : tm.\ \lambda a : tm \to Prop.\ Prf\ (a\ x)$

From SFOL to UFOL

- **Intuition**: transforming sorts into sort predicates

$\mu(Set) = tm \to Prop$ $\qquad\qquad$ $\mu(\#) = \lambda x : tm.\ \lambda a : tm \to Prop.\ Prf\ (a\ x)$

- **Problem**: we need a proof of the sort predicate

$\mu(\forall) = \lambda a : tm \to Prop.\ \lambda p : (\Pi x : tm.\ Prf\ (a\ x) \to Prop).\ \forall\ (\lambda x : tm.\ (a\ x) \Rightarrow (p\ x\ ?))$

From SFOL to UFOL

- **Intuition**: transforming sorts into sort predicates

  $\mu(Set) = tm \to Prop$ $\qquad$ $\mu(\#) = \lambda x : tm.\ \lambda a : tm \to Prop.\ Prf\ (a\ x)$

- **Problem**: we need a proof of the sort predicate

  $\mu(\forall) = \lambda a : tm \to Prop.\ \lambda p : (\Pi x : tm.\ Prf\ (a\ x) \to Prop).\ \forall\ (\lambda x : tm.\ (a\ x) \Rightarrow (p\ x\ ?))$

- **Solution**: we use a **dependent implication**

  $Prf\ (p \Rightarrow_d q) \hookrightarrow \Pi h : Prf\ p.\ Prf\ (q\ h)$

  $\mu(\forall) = \lambda a : tm \to Prop.\ \lambda p : (\Pi x : tm.\ Prf\ (a\ x) \to Prop).\ \forall\ (\lambda x : tm.\ (a\ x) \Rightarrow_d (\lambda h.\ p\ x\ h))$

From SFOL to UFOL

- **Intuition**: transforming sorts into sort predicates

  $\mu(Set) = tm \to Prop$ $\qquad\qquad \mu(\#) = \lambda x : tm.\ \lambda a : tm \to Prop.\ Prf\ (a\ x)$

- **Problem**: we need a proof of the sort predicate

  $\mu(\forall) = \lambda a : tm \to Prop.\ \lambda p : (\Pi x : tm.\ Prf\ (a\ x) \to Prop).\ \forall\ (\lambda x : tm.\ (a\ x) \Rightarrow (p\ x\ ?))$

- **Solution**: we use a **dependent implication**

  $Prf\ (p \Rightarrow_d q) \hookrightarrow \Pi h : Prf\ p.\ Prf\ (q\ h)$

  $\mu(\forall) = \lambda a : tm \to Prop.\ \lambda p : (\Pi x : tm.\ Prf\ (a\ x) \to Prop).\ \forall\ (\lambda x : tm.\ (a\ x) \Rightarrow_d (\lambda h.\ p\ x\ h))$

- We have $\mu(Prf\ (\forall\ a\ p)) \equiv_{\beta\mathcal{R}} \mu(\Pi x : tm.\ \Pi h : x \# a.\ Prf\ (p\ x\ h))$ in UFOL

Outline

Implementation in Dedukti

- The user specifies the source file, the target file and the name of the output

Implementation in Dedukti

- The user specifies the source file, the target file and the name of the output
- The output file is generated and the user must **fill in the parameters**
  - The conditions on constants are checked automatically
  - The conditions on rewrite rules have to be checked by the user

Implementation in Dedukti

- The user specifies the source file, the target file and the name of the output
- The output file is generated and the user must **fill in the parameters**
  - The conditions on constants are checked automatically
  - The conditions on rewrite rules have to be checked by the user
- All the **results and proofs** of the source theory are now expressed in the target theory

In practice

- Available online

`https://github.com/Deducteam/TranslationTemplates`

In practice

- Available online

  `https://github.com/Deducteam/TranslationTemplates`

- All the examples have been implemented in Dedukti

Outline

Takeaway message

- **Theory morphisms** for the $\lambda\Pi$-calculus modulo rewriting
  - Implemented in Dedukti to easily **transfer proofs** between theories
  - Rewrite rules in the target theory: simplify the proof obligations
  - Rewrite rules in the source theory: create additional constraints

Takeaway message

- **Theory morphisms** for the $\lambda\Pi$-calculus modulo rewriting
  - Implemented in Dedukti to easily **transfer proofs** between theories
  - Rewrite rules in the target theory: simplify the proof obligations
  - Rewrite rules in the source theory: create additional constraints

- Technical features (dependent pairs and dependent implications) may be necessary

Takeaway message

- **Theory morphisms** for the $\lambda\Pi$-calculus modulo rewriting
  - Implemented in Dedukti to easily **transfer proofs** between theories
  - Rewrite rules in the target theory: simplify the proof obligations
  - Rewrite rules in the source theory: create additional constraints

- Technical features (dependent pairs and dependent implications) may be necessary

- Potential application: translation between the encodings of **B** and **TLA+** in Dedukti

Takeaway message

- **Theory morphisms** for the $\lambda\Pi$-calculus modulo rewriting
  - Implemented in Dedukti to easily **transfer proofs** between theories
  - Rewrite rules in the target theory: simplify the proof obligations
  - Rewrite rules in the source theory: create additional constraints

- Technical features (dependent pairs and dependent implications) may be necessary

- Potential application: translation between the encodings of **B** and **TLA+** in Dedukti

Thank you!