Kuroda's Translation for the $\lambda\Pi$-Calculus Modulo Theory and Dedukti

LFMTP 2024

Thomas Traversié

Interoperability between proof systems

- Many different proof systems

- Need for **interoperability**
  ↪ Re-usability, re-checking, preservation of databases

- The $\lambda\Pi$-calculus modulo theory [Cousineau and Dowek, 2007]
  - λ-calculus extended with dependent types and rewrite rules
  - Logical framework used for **proof exchange**
  - Implemented in the Dedukti proof language

Classical logic and intuitionistic logic

- Classical proof systems: HOL Light, Mizar
  Intuitionistic proof systems: Coq, Agda

- **Intuitionistic** logic = classical logic without the principle of the excluded middle $A \vee \neg A$

- Drawbacks:
  - No double-negation elimination $\neg\neg A \Rightarrow A$
  - No proof by contradiction

$$\frac{\Gamma, \neg A \vdash \bot}{\Gamma \vdash A}$$

- Advantage: **constructive** proofs

Embedding classical logic into intuitionistic logic

- Translations $A \mapsto A^*$ [Kolmogorov, 1925, Gödel, 1933, Gentzen, 1936, Kuroda, 1951]
  - Insert **double negations** inside formulas
  - In first-order logic,

$$\underbrace{\Gamma \vdash_c A}_{\text{classical logic}} \quad \text{iff} \quad \underbrace{\Gamma^* \vdash_i A^*}_{\text{intuitionistic logic}}$$

- Intuition:

$$\vdash_i A \vee \neg A \quad ✗$$

$$\vdash_i \neg\neg(A \vee \neg A) \quad ✓$$

- Kuroda's translation can be extended to **higher-order logic** [Brown and Rizkallah, 2014]

Contribution

- We **characterize** theories encoded in higher-order logic in the $\lambda\Pi$-calculus modulo theory
- We extend Kuroda's translation to the **$\lambda\Pi$-calculus modulo theory**
- We **implement** it for Dedukti proofs

Outline

Outline

Higher-order logic in the $\lambda\Pi$-calculus modulo theory

Kuroda's Translation for the $\lambda\Pi$-calculus modulo theory

Implementation for Dedukti proofs

Conclusion

The $\lambda\Pi$-calculus modulo theory

- Syntax

Sorts	$s ::= \mathtt{TYPE} \mid \mathtt{KIND}$
Terms	$t, u, A, B ::= c \mid x \mid s \mid \Pi x : A.\ B \mid \lambda x : A.\ t \mid t\ u$
Signatures	$\Sigma ::= \langle\rangle \mid \Sigma, c : A$
Rewrite systems	$\mathcal{R} ::= \langle\rangle \mid \mathcal{R}, \ell \hookrightarrow r$
Contexts	$\Gamma ::= \langle\rangle \mid \Gamma, x : A$

$\Pi x : A.\ B$ written $A \to B$ if x not in B

- Theory $\mathcal{T} = (\Sigma, \mathcal{R})$

- **Conversion** $\equiv_{\beta\mathcal{R}}$ generated by β-reduction and $\mathcal{R}$

Typing rules

$$\frac{\Gamma \vdash A : \texttt{TYPE} \qquad \Gamma, x : A \vdash B : s \qquad \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.\ t : \Pi x : A.\ B}\ [\textsc{Abs}]$$

$$\frac{\Gamma \vdash t : \Pi x : A.\ B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B[x \mapsto u]}\ [\textsc{App}]$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash B : s}{\Gamma \vdash t : B}\ [\textsc{Conv}]\ A \equiv_{\beta\mathcal{R}} B$$

Basic encoding

- Notions of proposition and proof [Blanqui et al, 2023]
- Universe of **sorts** $Set : \mathtt{TYPE}$, injection $El : Set \to \mathtt{TYPE}$
  Sort nat : Set, natural number $n : El$ nat
- Universe of **propositions** $Prop : \mathtt{TYPE}$, injection $Prf : Prop \to \mathtt{TYPE}$
  Proposition $P : Prop$, a proof of P is of type $Prf\ P$

Encoding connectives and quantifiers

- Encoding the **connectives** and **quantifiers** [Blanqui et al, 2023]

$$\vee : Prop \to Prop \to Prop$$

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

- Polymorphic quantifiers $\forall$ and $\exists$ over sorts

- **Higher-order encoding**
  – Sort of propositions $o : Set$, with $El\ o \hookrightarrow Prop$
  – Functionality $\leadsto$, with $El\ (x \leadsto y) \hookrightarrow El\ x \to El\ y$

$$\frac{\Gamma \vdash P \vee Q \qquad \Gamma, P \vdash R \qquad \Gamma, Q \vdash R}{\Gamma \vdash R}\ \textsc{Or-E}$$

$$\begin{aligned} \mathsf{or_e} : \quad & \Pi p, q : Prop. \\ & Prf\ (p \vee q) \to \\ & \Pi r : Prop. \\ & (Prf\ p \to Prf\ r) \to \\ & (Prf\ q \to Prf\ r) \to \\ & Prf\ r \end{aligned}$$

Characterizing higher-order logic (1)

- Signatures Σ^i_{HOL} for **intuitionistic** logic and Σ^c_{HOL} for **classical** logic

$$\Sigma^c_{HOL} = \Sigma^i_{HOL}, \mathrm{pem} : \Pi p : Prop.\ Prf\ (p \vee \neg p)$$

  Rewrite system $\mathcal{R}_{HOL}$

- User-defined constants $\Sigma_{\mathcal{T}}$ and rewrite rules $\mathcal{R}_{\mathcal{T}}$
  – We can **mix sorts, propositions and proofs**

$$c : \Pi P : Prop.\ Prf\ P \to El\ \mathrm{nat}$$

  – We must restrict the typed constants $\Sigma_{\mathcal{T}}$

Characterizing higher-order logic (2)

- Hierarchy

$$\kappa_1 ::= \mathit{Set} \mid \kappa_1 \to \kappa_1$$
$$\kappa_2 ::= \mathit{Prop} \mid \mathit{El}\ a \mid \Pi x : \kappa_i.\ \kappa_2 \text{ with } i \in \{1, 2\}$$
$$\kappa_3 ::= \mathit{Prf}\ p \mid \kappa_3 \to \kappa_3 \mid \Pi x : \kappa_i.\ \kappa_3 \text{ with } i \in \{1, 2\}$$
$$\kappa_4 ::= \mathtt{TYPE} \mid \Pi x : \kappa_i.\ \kappa_4 \text{ with } i \in \{1, 2\}$$
$$\kappa_5 ::= \mathtt{KIND}$$

- κ_3 represents formulas and inference rules

- **Constraint**: for every $c : A \in \Sigma_{\mathcal{T}}$, we have $A \in \kappa_i$ for some $i \in [\![1, 5]\!]$

Theories encoded in higher-order logic

- Theories **encoded** in higher-order logic $\mathcal{T} = (\Sigma^k_{HOL} \cup \Sigma_{\mathcal{T}}, \mathcal{R}_{HOL} \cup \mathcal{R}_{\mathcal{T}})$ with $k \in \{i, c\}$

- Example: arithmetic

$$
\begin{array}{lll}
\mathsf{nat} & : & \mathit{Set} \\
0 & : & \mathit{El}\ \mathsf{nat} \\
\mathsf{succ} & : & \mathit{El}\ \mathsf{nat} \to \mathit{El}\ \mathsf{nat} \\
+ & : & \mathit{El}\ \mathsf{nat} \to \mathit{El}\ \mathsf{nat} \to \mathit{El}\ \mathsf{nat}
\end{array}
\qquad
\begin{array}{lll}
x + 0 & \hookrightarrow & x \\
x + \mathsf{succ}\ y & \hookrightarrow & \mathsf{succ}\ (x + y)
\end{array}
$$

$$\mathsf{rec} : \mathit{Prf}\ (\forall\ (\mathsf{nat} \leadsto o)\ (\lambda P.\ (P\ 0 \land (\forall\ \mathsf{nat}\ (\lambda n.\ P\ n \Rightarrow P\ (\mathsf{succ}\ n)))) \Rightarrow (\forall\ \mathsf{nat}\ (\lambda n.\ P\ n))))$$

Outline

Higher-order logic in the $\lambda\Pi$-calculus modulo theory

Kuroda's Translation for the $\lambda\Pi$-calculus modulo theory

Implementation for Dedukti proofs

Conclusion

Kuroda's translation...

- **Principle of the translation**: inserting **double negations** in front of formulas and after every universal quantifier

- Challenges in the encoding of higher-order logic in the $\lambda\Pi$-calculus modulo theory
  - Dependent types
  - Rewrite rules
  - Proofs are terms

- **Principle of the proof**: the translation of each natural deduction rule is **admissible** in **intuitionistic** logic

$$\frac{\Gamma, P \vdash_c Q}{\Gamma \vdash_c P \Rightarrow Q}\ \textsc{Imp-I} \qquad\qquad \frac{\Gamma^{Ku}, P^{Ku} \vdash_i Q^{Ku}}{\Gamma^{Ku} \vdash_i (P \Rightarrow Q)^{Ku}}$$

- For each constant $c : A \in \Sigma_{HOL}$ representing a natural deduction rule, we build a **term** c^i of type A^{Ku} in $(\Sigma^i_{HOL}, \mathcal{R}_{HOL})$

Kuroda's translation in the $\lambda\Pi$-calculus modulo theory (1)

- Translation of **terms**

$$x^{Ku} := x \qquad (\lambda x : A.\ t)^{Ku} := \lambda x : A^{Ku}.\ t^{Ku}$$

$$(t\ u)^{Ku} := t^{Ku}\ u^{Ku} \qquad (\Pi x : A.\ B)^{Ku} := \Pi x : A^{Ku}.\ B^{Ku}$$

$$c^{Ku} := \begin{cases} \lambda p.\ Prf\ (\neg\neg p) & \text{if } c = Prf \\ \lambda x.\ \lambda p.\ \forall\ x\ (\lambda z.\ \neg\neg(p\ z)) & \text{if } c = \forall \\ c^{i} & \text{if } c \text{ represents a natural deduction rule} \\ c & \text{otherwise} \end{cases}$$

- Substitution: $(t[z \leftarrow w])^{Ku} = t^{Ku}[z \leftarrow w^{Ku}]$

Kuroda's translation in the $\lambda\Pi$-calculus modulo theory (2)

- Translation of contexts, signatures and rewrite systems

$$\begin{aligned} \langle\rangle^{Ku} &::= \langle\rangle \\ (\Gamma, x : A)^{Ku} &:= \Gamma^{Ku}, x : A^{Ku} \\ (\Sigma, c : A)^{Ku} &:= \Sigma^{Ku}, c : A^{Ku} \\ (\mathcal{R}, \ell \hookrightarrow r)^{Ku} &:= \mathcal{R}^{Ku}, \ell^{Ku} \hookrightarrow r^{Ku} \end{aligned}$$

- Translation of **theory** $\mathcal{T} = (\Sigma^c_{HOL} \cup \Sigma_{\mathcal{T}}, \mathcal{R}_{HOL} \cup \mathcal{R}_{\mathcal{T}})$

$$\mathcal{T}^{Ku} = (\Sigma^i_{HOL} \cup \Sigma^{Ku}_{\mathcal{T}}, \mathcal{R}_{HOL} \cup \mathcal{R}^{Ku}_{\mathcal{T}})$$

- Conversion: if $A \equiv_{\beta\mathcal{R}} B$ in $\mathcal{T}$ then $A^{Ku} \equiv_{\beta\mathcal{R}} B^{Ku}$ in $\mathcal{T}^{Ku}$

- **Theorem**: If $\Gamma \vdash t : A$ in $\mathcal{T}$ then $\Gamma^{Ku} \vdash t^{Ku} : A^{Ku}$ in $\mathcal{T}^{Ku}$

- Every occurrence of the **classical** axiom

$$\mathrm{pem} : \Pi p : Prop.\ Prf\ (p \vee \neg p)$$

is replaced by the **intuitionistic** proof term

$$\mathrm{pem}^{i} : \Pi p : Prop.\ Prf\ (\neg\neg(p \vee \neg p))$$

Back to the original theory

- **Theorem**: if $\Gamma^{Ku} \vdash t : A^{Ku}$ in $\mathcal{T}^{Ku}$ then there exists some term t' such that $\Gamma \vdash t' : A$ in $\mathcal{T}$

- Proof in two steps:
  - From a proof of A^{Ku}, build a proof of A in classical logic
  - From a proof of A that uses $\Sigma_{\mathcal{T}}^{Ku}$ and $\mathcal{R}_{\mathcal{T}}^{Ku}$, build a proof of A that uses $\Sigma_{\mathcal{T}}$ and $\mathcal{R}_{\mathcal{T}}$

Outline

Construkti

- Dedukti = a **proof language** for the $\lambda\Pi$-calculus modulo theory

- Construkti = a **tool** that implements Kuroda's translation
  - Takes a Dedukti file in classical logic
  - Outputs a Dedukti file in intuitionistic logic

- Example: Clavius law $\forall A\ (\neg A \Rightarrow A) \Rightarrow A$

```
thm clavius : Prf (all o (A : El o => (imp (imp (not A) A) A)))
```

```
thm clavius_i :
   Prf (not (not (all o (A : El o =>
                                  not (not (imp (imp (not A) A) A))))))
```

In practice

Construkti inserts double negations and replaces the **constants** $c : A$ representing the **classical** natural deduction rules by **intuitionistic proof terms** $c^i : A^{Ku}$

```
pem : p : Prop -> Prf (or p (not p)).
```

```
thm pem_i : p : Prop -> Prf (not (not (or p (not p))))
:= p => neg_i (not (or p (not p)))
    (pNPNP => neg_e (or p (not p)) pNPNP
        (or_ir p (not p)
            (neg_i p (pP => neg_e
                (or p (not p))
                pNPNP
                (or_il p pP (not p)))))).
```

Benchmark

- Tested on **100 proofs**
  - In propositional, first-order and higher-order logic
  - Provable formulas and admissible inference rules
  - Classical formulas, De Morgan's laws, polymorphic Leibniz equality, arithmetic
  - Using rewrite rules and dependent types

- Tool and benchmark available at

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

Outline

Higher-order logic in the $\lambda\Pi$-calculus modulo theory

Kuroda's Translation for the $\lambda\Pi$-calculus modulo theory

Implementation for Dedukti proofs

Conclusion

Takeaway message

- Kuroda's translation extends to theories encoded in higher-order logic in the $\lambda\Pi$-calculus modulo theory

- It is both:
  – an **extension** to a logical framework with dependent types and rewrite rules
  – an **encoding** inside a logical framework where proofs are terms

- Tool Construkti $=$ an **implementation** in Dedukti

Perspectives

- Dedukti and Construkti paves the way for **proof interoperability**
- Future work: apply Construkti on a large database of proofs
- Related work: **constructivisation** [Cauderlier, 2016, Gilbert, 2017]
  - Kuroda: *always* finds an intuitionistic proof, but *modifies* the theorem
  - Constructivisation: finds an intuitionistic proof for the *original* theorem, but *may fail*

Thank you for your attention!