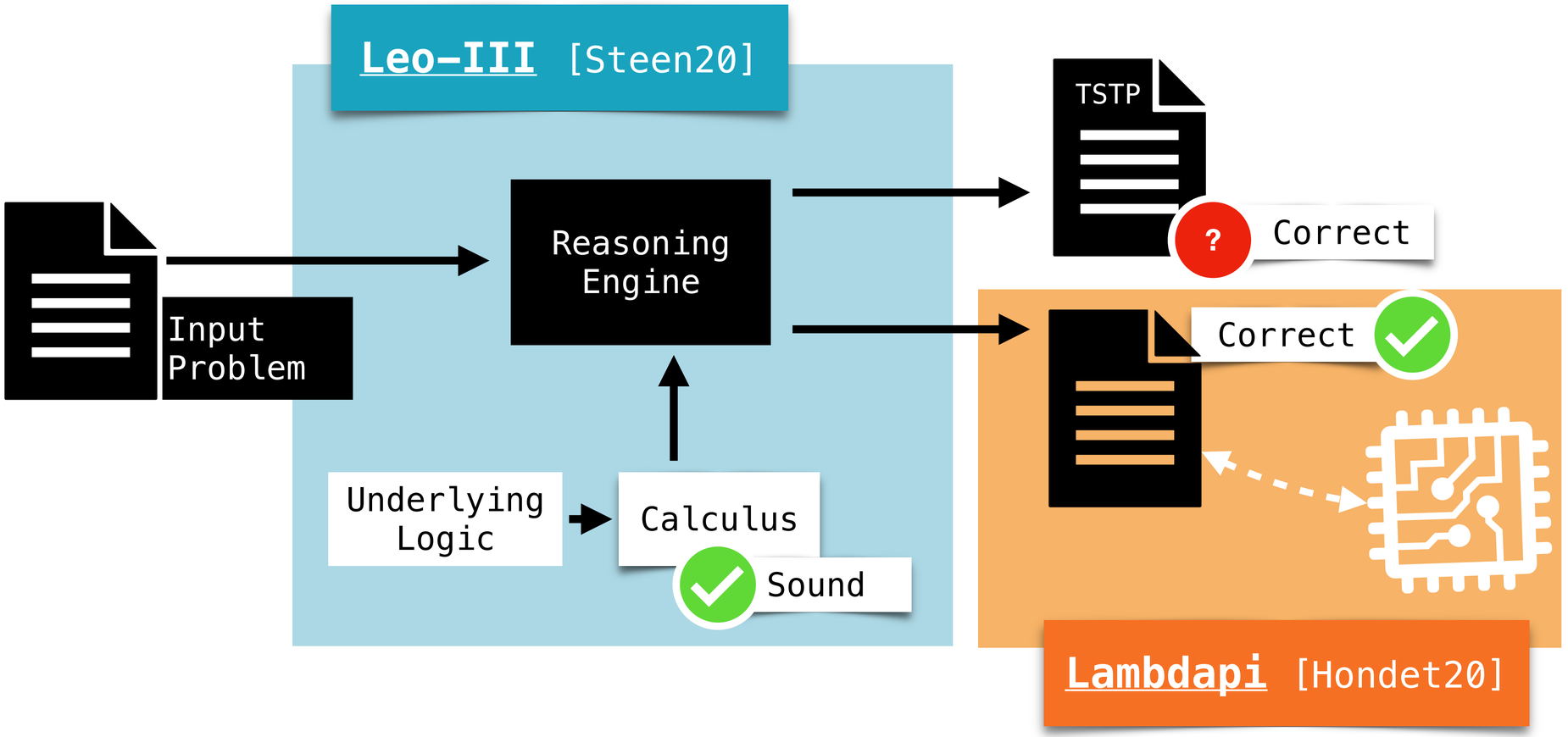
Towards the Verification of Higher-Order Logic Automated Reasoning



UNIVERSITÄT GREIFSWALD
Wissen lockt. Seit 1456

Melanie Taprogge

Supervised by:
Frédéric Blanqui
Alexander Steen

école normale supérieure paris-saclay

Verification in the Dedukti Framework [Assaf16]

- Goal: Encode and check proofs following the propositions as types principle
  - Curry-Howard Correspondence: [Curry34,Howard80]
    Proofs are encoded as terms, their types are the propositions they proof
  - Brouwer–Heyting–Kolmogorov interpretation: [Brouwer75,Heyting30,Kolmogoroff32]
    Logical connectives are treated as type constructors

Let B be a proposition. A term p_B then represents a proof of it.

$(A \Rightarrow B)$ is identified with $A \to B$

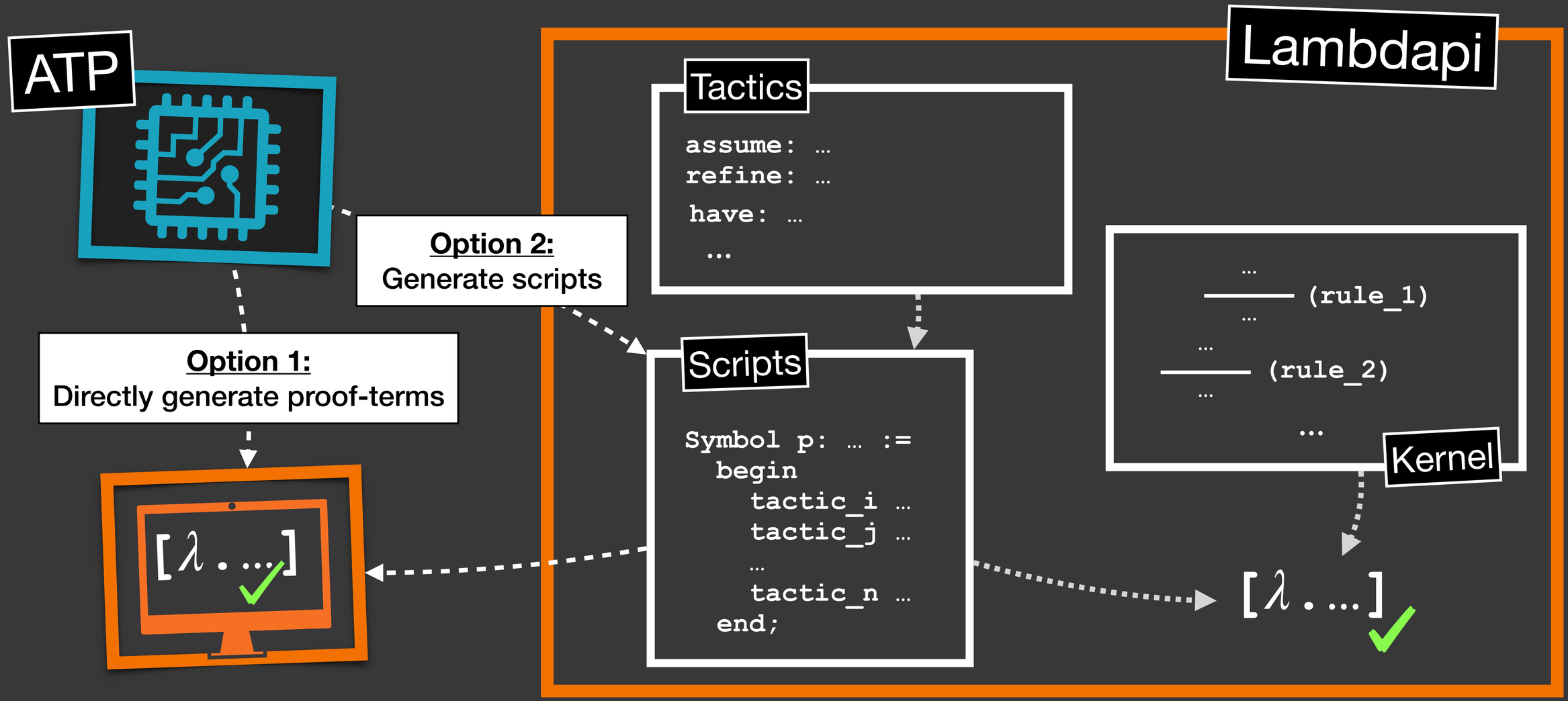
$q_{B \to A}\ p_B$ proofs A

$TYPE$ allows us to map propositions to types

Rewrite rules are responsible for the identification

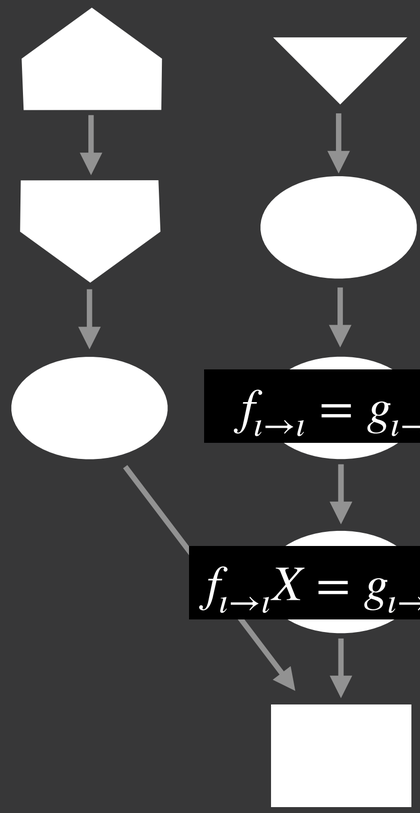
+ Dependant types: $\Pi x : T\,.\,S$, for instance $\Pi x : Nat\,.\,array(x)$
+ Meta-logical type of types (called $TYPE$) e.g: $array_{Nat \to TYPE}$
+ Rewrite rules $l \hookrightarrow r$ replace occurrences of their *LHS* with the term of their *RHS*

Encoding Proofs in Lambdapi



Found Proof

Leo-III



$f_{\iota\to\iota} = g_{\iota\to\iota}$

$f_{\iota\to\iota}X = g_{\iota\to\iota}X$

1 Definition of a **Lambdapi Theory**

2 Encoding of **Problems and Proof** Steps

3 Encoding of the **Calculus Rules**

4 Verification of **generated Proofs**

PFE

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu}}{s_{\tau\to\nu}X_\tau = t_{\tau\to\nu}X_\tau}$$

Encoded Proof

Lambdapi

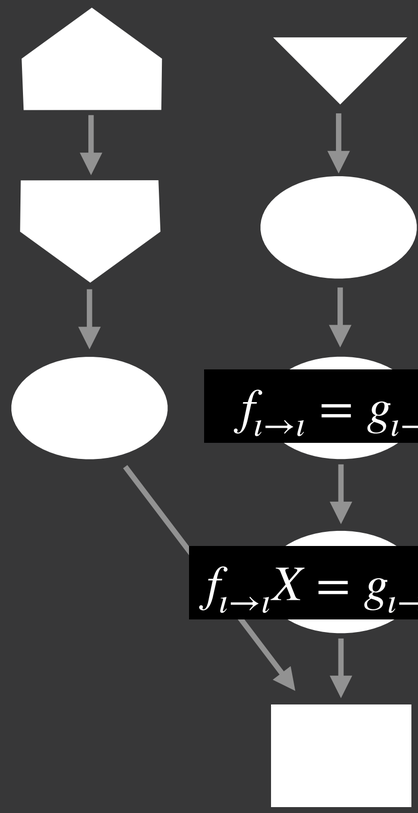
```
require open StdLib.… ;
…

symbol axiom_i : … ;
…

symbol proof: conjecture ≔
begin
    …

    have step_m: …
      {…};

    have step_n: …
      {…};

      …

end;
```

Found Proof

1 2

Leo-III

Extensional Type Theory
(ExTT) [Henkin50]
-> HOL + Extensionality
+ choice
+ Rank-1 Polymorphism



$f_{\iota\to\iota} = g_{\iota\to\iota}$

$f_{\iota\to\iota}X = g_{\iota\to\iota}X$

PFE

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu}}{s_{\tau\to\nu}X_\tau = t_{\tau\to\nu}X_\tau}$$

Encoded Proof

Lambdapi

$\lambda\Pi$-calculus modulo
-> HOL + dependant types
+ rewriting

Encodings of the Standard Library [Blanqui23]

Propositions as types!

OL Types

```
Set : TYPE
ι : Set
~ : Set → Set → Set
```

Objects

```
τ : Set → TYPE
f : τ (ι ~ ι)
```

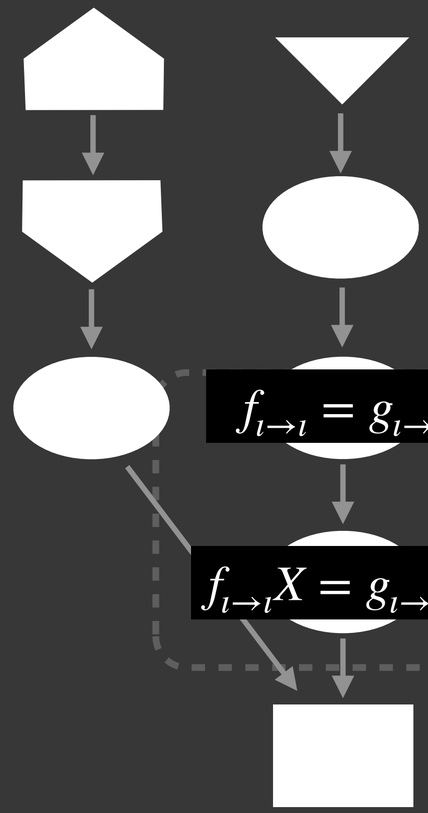
Formulas

```
step_m : π (f = g)
step_n : Π (x : τ ι),
         π (g x = f x)
π : τ o → TYPE
```

Found Proof

1 2 3

Leo-III



Calculus EP [Steen20]
(a calculus for Extensional Higher-Order Paramodulation)

PFE

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu}}{s_{\tau\to\nu}X_\tau = t_{\tau\to\nu}X_\tau}$$

$f_{\iota\to\iota} = g_{\iota\to\iota}$

$f_{\iota\to\iota}X = g_{\iota\to\iota}X$

Encoded Proof

Lambdapi

$\lambda\Pi$-calculus modulo
-> HOL + dependant types
+ rewriting

Propositions as types!

```
step_m : π (f = g)
```

PFE

```
step_n : Π (x : τ ι), π (g x = f x)
```

```
have step_m : π (f = g)
{…};

have step_n : Π (x : τ ι),
  π (f x = g x)
  {
    assume x;

    refine PFE ι ι f g x step_m

  };
```

Found Proof

1 2 3 4

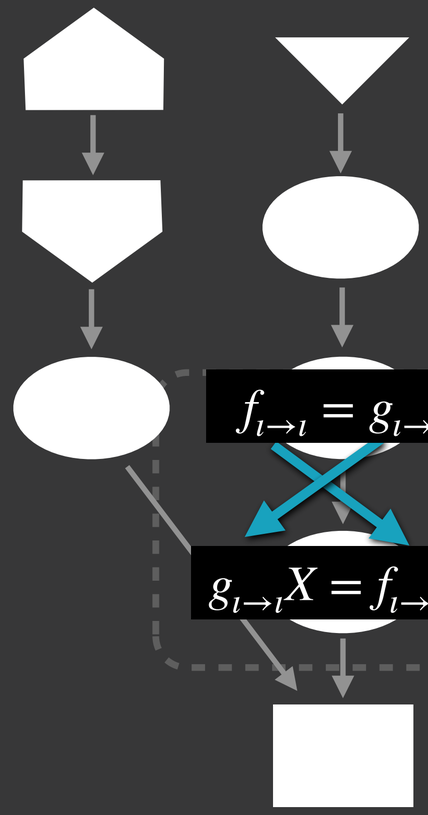
Encoded Proof

Leo-III

Lambdapi

$\lambda\Pi$-calculus modulo
-> HOL + dependant types
+ rewriting

Propositions as types!



Calculus EP [Steen20]
(a calculus for Extensional Higher-Order Paramodulation)

$f_{\iota\to\iota} = g_{\iota\to\iota}$

$g_{\iota\to\iota}X = f_{\iota\to\iota}X$

PFE

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu}}{s_{\tau\to\nu}X_\tau = t_{\tau\to\nu}X_\tau}$$

```
step_m : π (f = g)
```

PFE

```
step_n : Π (x : τ ι), π (g x = f x)
```

```
have step_m : π (f = g)
{…};

have step_n : Π (x : τ ι),
  π (f x = g x)
  {
    assume x;

    refine PFE ι ι f g x step_m

  };
```

Found Proof

1 2 3 4

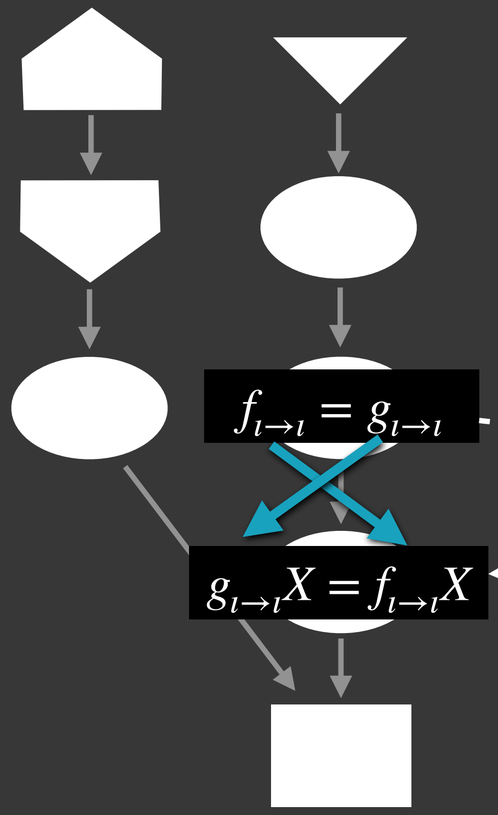
Encoded Proof

Leo-III

Lambdapi

λΠ-calculus modulo
-> HOL + dependant types
+ rewriting

Propositions as types!



$f_{\iota\to\iota} = g_{\iota\to\iota}$

$g_{\iota\to\iota}X = f_{\iota\to\iota}X$

PFE

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu}}{s_{\tau\to\nu}X_\tau = t_{\tau\to\nu}X_\tau}$$

```
step_m : π (f = g)
```

PFE

```
Π (x : τ ι), π (g x = f x)
```

= _sym

```
step_n : Π (x : τ ι), π (f x = g x)
```

$$\frac{s = t}{t = s}$$

Reordering of terms

Simplifications

…

Identify possible additional modifications, encode as inference rules

When verifying a step in a proof, detect all actually occurring modifications and apply corresponding rules in a modular fashion

Found Proof

1 2 3 4

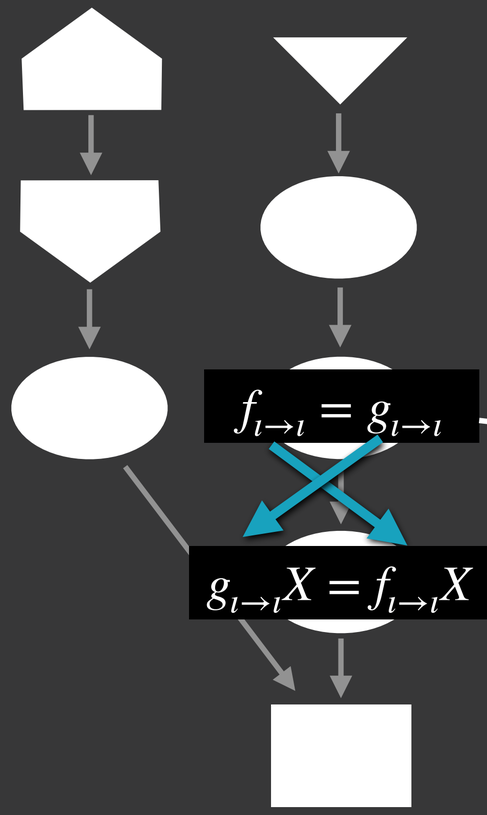
Encoded Proof

Leo-III

Lambdapi

λΠ-calculus modulo
-> HOL + dependant types
+ rewriting

Propositions as types!



PFE

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu}}{s_{\tau\to\nu}X_\tau = t_{\tau\to\nu}X_\tau}$$

```
have step_m : π (f = g)
{…};

have step_n : Π (x : τ ι), π (g x = f x)
  {
   assume x;
   have PFEapp: π (f x = g x)
      {refine PFE ι ι f g x step_m};
   refine =sym ι (f x) (g x) PFEapp
  };
```

Calculus Rule

Implicit Transformation

Found Proof

Leo-III

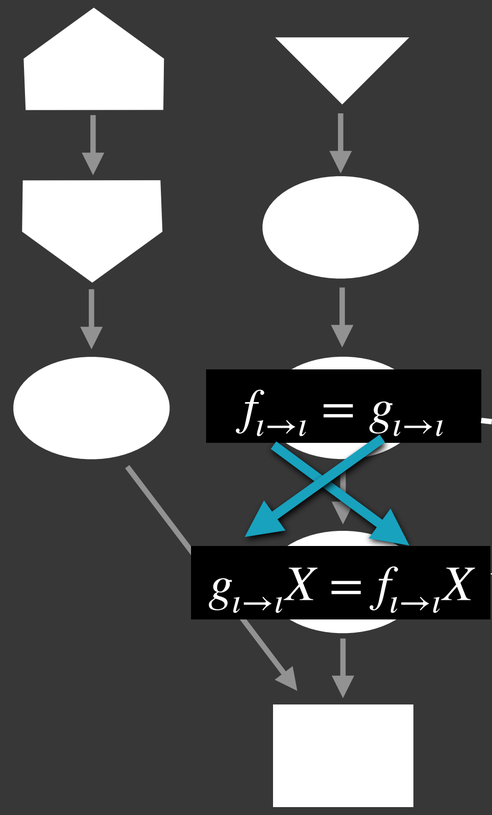
1 2 3 4

Encoded Proof

Lambdapi

$\lambda\Pi$-calculus modulo
-> HOL + dependant types
+ rewriting

Propositions as types!



PFE

$$\frac{s_{\tau\to\nu} = t_{\tau\to\nu}}{s_{\tau\to\nu}X_\tau = t_{\tau\to\nu}X_\tau}$$

Encoding/ Application of rules can be more complicated:

Rules can apply only to substructures of clauses, e.g :

$$\frac{C \vee [s_{\tau\to\nu} = t_{\tau\to\nu}]}{C \vee [s_{\tau\to\nu}X_\tau = t_{\tau\to\nu}X_\tau]}$$

Rules can require a higher degree of structural flexibility, e.g :

$$\frac{l_0 \vee l_1 \vee \ldots \vee l_n}{l_{\sigma(0)} \vee l_{\sigma(1)} \vee \ldots \vee l_{\sigma(n)}}$$

-> use of more intricate encodings, additional Lambdapi theorems or tactics for easier application

Progress

Evaluate current encoding and possibly change to more efficient strategies

○ Extend to Polymorphism

⦸ Detect and encode all implicit Transformations

⦸ Change Leo-III to avoid unnecessary transformations

1 Definition of a Lambdapi Theory

2 Encoding of Problems and Proof Steps

3 Encoding of the Calculus Rules

4 Verification of generated Proofs

⦸ Include additional information in currently generated Leo-III proof objects

⊠ Extend Standard Library to encode monomorphic ExTT

⊠ Representing all (25) relevant EP Rules in Lambdapi

Integration into GDV

Pleasant side-effect: Discovery of minor bugs

Implementation currently for 13 Rules!
Coverage of verifiable rules on monomorphic HOL Benchmark : 40%

References

- Assaf, A., Burel, G., Cauderlier, R., Delahaye, D., Dowek, G., Dubois, C., ... & Saillard, R. (2016). Dedukti: a logical framework based on the λΠ-calculus modulo theory.
- Blanqui, Frédéric, et al. "A modular construction of type theories." *Logical Methods in Computer Science* 19 (2023).
- Cousineau, D., & Dowek, G. (2007). Embedding pure type systems in the lambda-pi-calculus modulo. In Typed Lambda Calculi and Applications: 8th International Conference, TLCA 2007, Paris, France, June 26-28, 2007. Proceedings 8 (pp. 102-117). Springer Berlin Heidelberg.
- Curry, H. B. (1934). Functionality in combinatory logic. *Proceedings of the National Academy of Sciences*, *20*(11), 584-590.
- Henkin, Leon. "Completeness in the theory of types1." *The Journal of Symbolic Logic* 15.2 (1950): 81-91.
- Heyting, Arend. "Die formalen Regeln der intuitionistischen Logik." Sitzungsbericht PreuBische Akademie der Wissenschaften Berlin, physikalisch-mathematische Klasse II (1930): 42-56.
- Hondet, Gabriel, and Frédéric Blanqui. "The new rewriting engine of dedukti." *arXiv preprint arXiv:2010.16115* (2020).
- Howard, W. A. (1980). The formulae-as-types notion of construction. To HB Curry: essays on combinatory logic, lambda calculus and formalism, 44, 479-490.
- Kolmogoroff, Andrej. "Zur deutung der intuitionistischen logik." Mathematische Zeitschrift 35.1 (1932): 58-65.
- Steen, A. (2020). Extensional paramodulation for higher-order logic and its effective implementation Leo-III. *KI-Künstliche Intelligenz*, *34*(1), 105-108.