Interoperability between arithmetic proofs using Dedukti

Gilles Dowek, Stéphane Graham-Lengrand, François Thiré

Interoperability between arithmetic proofs using Dedukti

François Thiré

DEDUC⊢EAM (INRIA)

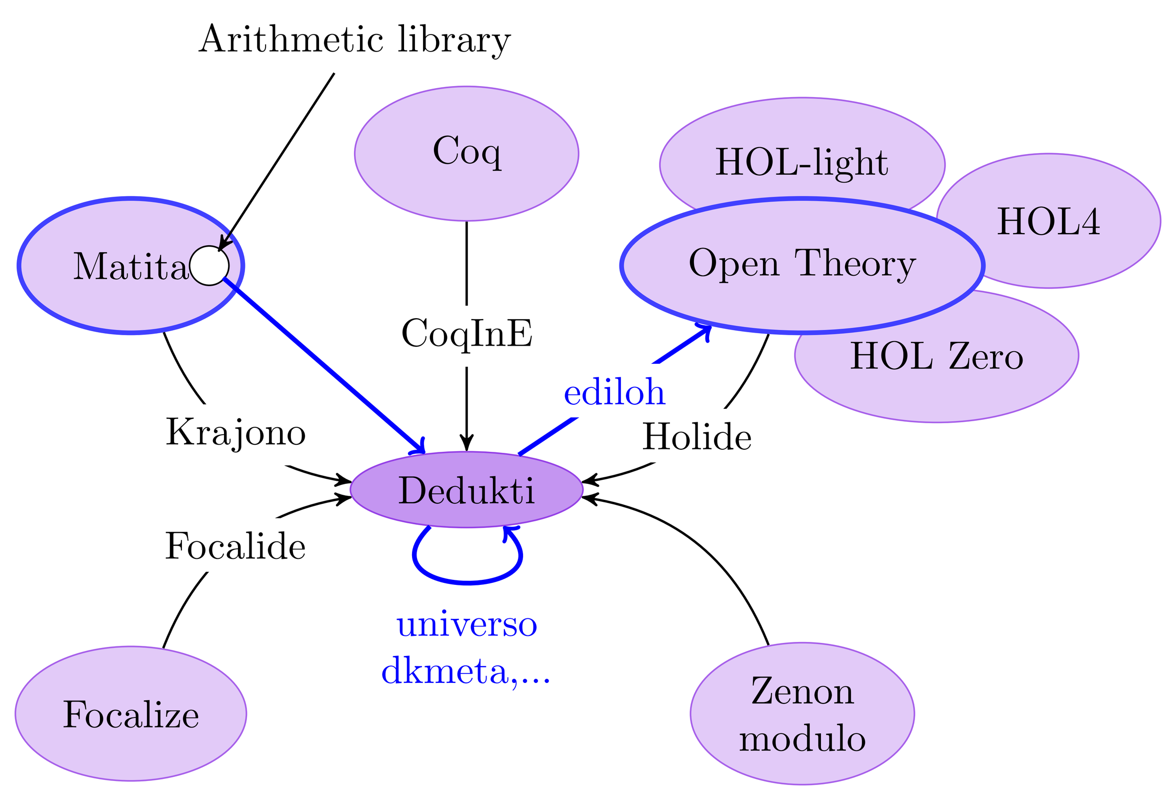
francois.thire@inria.fr

Motivations & objectives

- There exist many logics and many proof checkers for these logics
- Some logics are more *powerful* than others (e.g. quantify over proofs)
- Theorems and proofs are not shared between different proof checkers : Well-known theorems and proofs are proved **manually** for each new proof checker/logic
- There is **no standard** for these logics
- Our **objective** is to translate *automatically* a small library of arithmetic proofs from an expressive logic called *The Calculus of Inductive Constructions* (CiC), to a less expressive logic, *The Higher Order Logic* (HOL).

Dedukti is handy for interoperability

- DEDUKTI implements the $\lambda\Pi$-calcul modulo theory. It is a **logical framework**
- Logical frameworks are a kind of proof checker that allows to embed several logics
- Logical framework are good systems to make interoperability easier
- $\lambda\Pi$-calcul modulo theory is a **simple** logic that combines dependent types and rewrite rules
- The substantial advantage of dedukti for interoperability is that the encoding of a logical system $\mathcal{L}$ to dedukti is *shallow* :
  – Use of **Higher-order Abstract syntax**
  – **Type** preservation : $\Gamma \vdash_{\mathcal{L}} t : T \Rightarrow |\Gamma| \vdash_{\lambda\Pi} ||t|| : |T|$
  – **Computation** preservation : $t_1 \rightarrow_{\mathcal{L}} t_2 \Rightarrow ||t_1|| \rightarrow_{\lambda\Pi} ||t_2||$



Dedukti[CiC] to Dedukti[HOL]

- This translation is not always possible : In CiC, it is possible to quantify over proofs, not in HOL or in CiC, there is an infinite hierarchy of universes that does not exist in HOL
- But there should be a translation for arithmetic proofs : one does not need universes nor to quantify over proofs

Features to remove :
- Universes
- Dependent types
- Inductive definitions and recursive definitions (encoded in $\lambda\Pi$-calcul modulo theory by rewrite rules)

```
nat : Type.

0 : nat.
S : nat -> nat.
[] one --> S 0.

odd : nat -> Prop.
pi : odd (S 0).

def pi1 : (odd one) := pi.
```

With rewrite rules

```
[...]

def eq : nat -> nat -> Prop.
[x,y] eq x y -->
      forall (P:(nat -> Prop) =>
        impl (P x) (P y)).

eq_one : eq (S 0) one.
def pi1 : (odd one) :=
    eq_one (ctx => odd ctx) pi.
```

Without rewrite rules

Dedukti as a proof assistant

- Implements tactic in Dedukti (R. Cauderlier)
- Use external provers to prove intermediate results (A. Defourné)
- Prove that the convertibility test is decidable by proving the termination (G. Genestier)
- Implicits, Elaboration and Unification in Dedukti (R. Bocquet)

Dedukti[HOL] to OpenTheory

- Open Theory (OT) is already a tool for interoperability between *HOL family* provers.
- $\forall$ and $\Rightarrow$ are primitives in Dedukti[HOL] but not in OT. This extends to logical rules like the *Modus Ponens*
- Terms in Dedukti[HOL] are modulo β but not in OT where this conversion is explicit.
- Other technical problems arise such as that Dedukti uses De Bruijn indices but OT does not, polymorphism in Dedukti[HOL] does not work the same way as in OT...

Implementation

- Implementation in OCaml (5000 lines)
- Compilation time with Ediloh (Fermat little theorem) : 20s
- Several independent tools :
  – Universo (remove universes)
  – Deduktipli (remove dependent types)
  – Ediloh (the compiler from Dedukti[HOL] to OpenTheory)

Future Work

- Automatize the translation
- Extend that arithmetic library to other provers such as Coq, Matita, PVS...
- Create the W3P (W3C of proofs) in order to create the first standards for proofs
- OCaml is not really handy to write such compilers especially for binders. A joint work with Prof. Brigitte Pientka is to look at a new logical/programming system that would be handy to express proof transformations

- Embed other logical systems like CubicalTT (C. Leena Subramaniam)
- Extend Dedukti to rewrite modulo a congruence (G. Ferey)