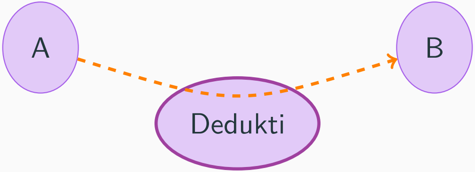
(Proof) Interoperability between proof systems with the Logical Framework Dedukti

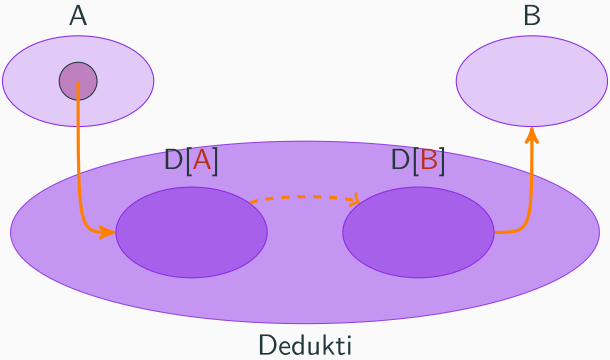
François Thiré

June 25, 2022

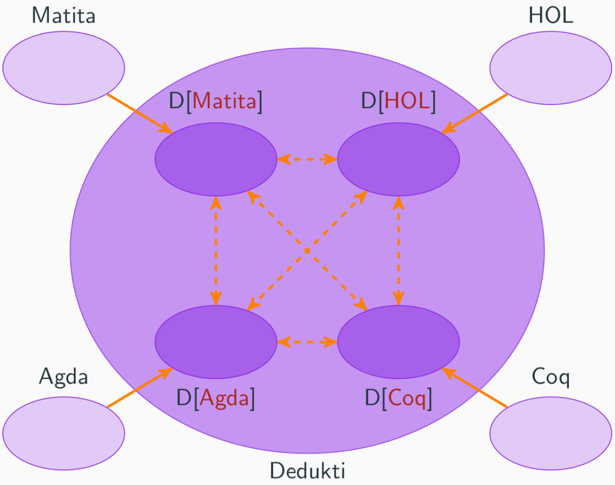
Nomadic Labs

Introduction


A
B
Dedukti


A
B
D[A]
D[B]
Dedukti

The quadratic problem reloaded?



What are the advantages of using Dedukti for interoperability?

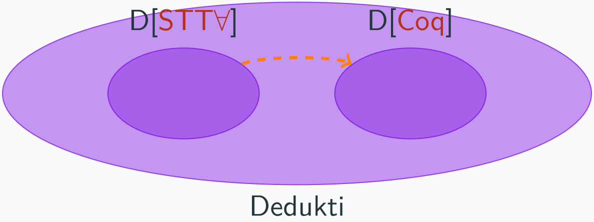
What are the advantages of using Dedukti for interoperability?

This is what we will try to answer during this lecture!

Setup for the demo

```
opam install dedukti
opam install universo
git clone https://github.com/Deducteam/Dedukti
git checkout francois@summer-school
cd Dedukti/summer-school
```

Objective of the demo



The logical framework Dedukti

Dedukti is a syntax for dependent types and rewriting

Dedukti syntax (1/2)

```
nat : Type.

0 : nat.

S : nat -> nat.

def plus : nat -> nat -> nat.

[m]   plus 0     m --> m.
[n,m] plus (S n) m --> S (plus n m).
```

Dedukti syntax (2/2)

```
(; Vector of singletons. ;)
vec : nat -> Type.
nil : vec 0.
cons : (n : nat) -> vec n -> vec (S n).

def append : (n : nat) -> (m : nat) -> vec n -> vec m -> vec (plus n m).
[r]       append _ _ nil r         --> r.
[n,m,l,r] append _ m (cons n l) r --> cons (plus n m) (append n m l r).

(; The rule below is also valid ;)
[n,m,l,r] append (S n) m (cons n l) r --> cons (plus n m) (append n m l r).
```

Table of contents

STT∀

High-level description of STT∀

A logic which features:

- Simply Type Lambda Calculus
- Prenex polymorphism (similar to OCaml polymorphism)
- Constructive, Impredicative and Higher-Order logic based on the quantifier $\forall$ (and its non dependent version $\Rightarrow$)

Shallow encodings with Dedukti

- Shallow vs deep is rather a spectrum with blur lines.
- For Dedukti, shallow generally means: a typing judgement of the source logic is translated into a typing judgement of Dedukti.
- shallow embeddings enable proof interoperability that scales

Demo

Let's try to understand the STT$\forall$ embedding and play with it.

Dkmeta

Dkmeta

Dkmeta is a tool to write term transformations with Dedukti

- Normalize a term according to a set of rewrite rules
- Dkmeta is implemented with the `dk` tool suite ($\approx$ 100 lines of OCaml code)

Purpose:

- Can be used to write many transformations (such as constant renaming)
- Can be used to write tactics in Dedukti

Example of use-case for dkmeta

$$Vec : \mathbb{N} \to \textbf{Type}$$

$$m : Vec\ 2$$

$$cons : (n : \mathbb{N}) \to Vec\ n \to Vec\ (n+1)$$

Example of use-case for dkmeta

$$Vec : \mathbb{N} \to \mathbf{Type}$$

$$m : Vec\ 2$$

$$cons : (n : \mathbb{N}) \to Vec\ n \to Vec\ (n+1)$$

$$plus : (x : \mathbb{N}) \to (y : \mathbb{N}) \to \mathbb{N}$$

Example of use-case for dkmeta

$$Vec : \mathbb{N} \to \textbf{Type}$$

$$m : Vec\ 2$$

$$cons : (n : \mathbb{N}) \to Vec\ n \to Vec\ (n+1)$$

$$plus : (x : \mathbb{N}) \to (y : \mathbb{N}) \to \mathbb{N}$$

We want to remove the unnecessary dependency:

$$plus : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$$

Removing dependent product (1/2)

How to write the following transformation in Dedukti?

```
plus : forall nat (x : Term (type z) nat =>
       forall nat (y : Term (type z) nat =>
              nat))
```

↓

```
plus : arr nat (arr nat nat)
```

Removing dependent product (1/2)

How to write the following transformation in Dedukti?

```
plus : forall nat (x : Term (type z) nat =>
       forall nat (y : Term (type z) nat =>
            nat))
```

$$\downarrow$$

```
plus : arr nat (arr nat nat)
```

With a usual programming language (Ocaml, Haskell, ...)

Code difficult to maintain because not resilient to changes!

- Hundred of lines of code to maintain
- The object logic evolves, alongside its encoding in Dedukti
- Depends on a specific implementation of Dedukti
- Each implementation of Dedukti aims to evolve

Removing dependent product (2/2)

Other idea: Use rewrite rules to do this transformation!

```
[A,F] forall A (x => F) --> arr A F.
```

Removing dependent product (2/2)

Other idea: Use rewrite rules to do this transformation!

```
[A,F] forall A (x => F) --> arr A F.
```

```
plus : forall nat (x : Term (type z) nat =>
       forall nat (y : Term (type z) nat =>
                   nat))
```

Removing dependent product (2/2)

Other idea: Use rewrite rules to do this transformation!

```
[A,F] forall A (x => F) --> arr A F.
```

```
plus : arr    nat
      (forall nat (y : Term (type z) nat =>
              nat))
```

Removing dependent product (2/2)

Other idea: Use rewrite rules to do this transformation!

```
[A,F] forall A (x => F) --> arr A F.
```

```
plus : arr    nat
      (arr    nat
              nat)
```

CTS: A parametric type theory

$$\mathscr{C} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{C})$$

Sorts

Cumul.

Axioms

Rules

Syntax

$$t, u, A, B ::= s \in \mathcal{S} \mid x \mid t\ u \mid \lambda x : A.\, t \mid (x : A) \to B$$

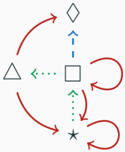
$$\frac{\Gamma \vdash_{\mathscr{C}} A : s_1 \qquad \Gamma, x : A \vdash_{\mathscr{C}} B : s_2 \qquad (s_1, s_2, s_3) \in \mathcal{R}}{\Gamma \vdash_{\mathscr{C}} (x : A) \to B : s_3}\ \mathscr{C}_{prod}$$

$$\frac{\Gamma \vdash_{\mathscr{C}} \mathbf{wf} \qquad (s_1, s_2) \in \mathcal{A}}{\Gamma \vdash_{\mathscr{C}} s_1 : s_2}\ \mathscr{C}_{sort} \qquad \frac{\Gamma \vdash_{\mathscr{C}} t : A \qquad \Gamma \vdash_{\mathscr{C}} B : s \qquad A \preceq^{\mathcal{C}}_{\mathscr{C}} B}{\Gamma \vdash_{\mathscr{C}} t : B}\ \mathscr{C}_{Conv}$$

Graph representation of a CTS

- $(s1, s2) \in \mathcal{A}$ is represented as $s_1 \dashrightarrow s_2$
- $(s1, s2) \in \mathcal{C}$ is represented as $s_1 \dashrightarrow s_2$
- $(s1, s2, s2) \in \mathcal{R}$ is represented as $s_1 \longrightarrow s_2$

STT∀ as a CTS



Demo

Let's use Dkmeta to go from the usual STT$\forall$ representation to its CTS representation.

Universo

Remember

$$\mathscr{C} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{C})$$

Sorts

Axioms

Rules

Cumul.

Syntax

$$t, u, A, B ::= s \in \mathcal{S} \;\mid\; x \;\mid\; t\;u \;\mid\; \lambda x : A.\, t \;\mid\; (x : A) \to B$$

$$\frac{\Gamma \vdash_{\mathscr{C}} A : s_1 \qquad \Gamma, x : A \vdash_{\mathscr{C}} B : s_2 \qquad (s_1, s_2, s_3) \in \mathcal{R}}{\Gamma \vdash_{\mathscr{C}} (x : A) \to B : s_3}\ \mathscr{C}_{prod}$$

$$\frac{\Gamma \vdash_{\mathscr{C}} \mathbf{wf} \qquad (s_1, s_2) \in \mathcal{A}}{\Gamma \vdash_{\mathscr{C}} s_1 : s_2}\ \mathscr{C}_{sort} \qquad \frac{\Gamma \vdash_{\mathscr{C}} t : A \qquad \Gamma \vdash_{\mathscr{C}} B : s \qquad A \preceq^{\mathcal{C}}_{\mathscr{C}} B}{\Gamma \vdash_{\mathscr{C}} t : B}\ \mathscr{C}_{Conv}$$

Universo

- Universo is about 1000 lines of OCaml
- Independent of the CTS specification
- Can be used to go from an impredicative theory to a predicative one
- Can be used to encode floating universes in Dedukti
- Can be used to minimize the number of universes needed
- Can be used to know whether some proofs can be encoded into another!

Paradox in Type Theory

$$\frac{}{\Gamma \vdash \mathit{Type} : \mathit{Type}} \quad ✗$$

Universes

$$\frac{}{\vdash U_i : U_{i+1}} \qquad \frac{\Gamma \vdash A : U_i \qquad \Gamma, x : A \vdash B : U_i}{\Gamma \vdash (x : A) \to B : U_i}$$

Universes

$$\frac{}{\vdash U_i : U_{i+1}} \qquad \frac{\Gamma \vdash A : U_i \qquad \Gamma, x : A \vdash B : U_i}{\Gamma \vdash (x : A) \to B : U_i}$$

Prop $:\equiv U_0$

Type $:\equiv U_1$

Kind $:\equiv U_2$

Universes

$$\frac{}{\vdash U_i : U_{i+1}} \qquad \frac{\Gamma \vdash A : U_i \qquad \Gamma, x : A \vdash B : U_i}{\Gamma \vdash (x : A) \to B : U_i}$$

$$nat : \textbf{Type}$$

$$nat \to nat : \textbf{Type}$$

$$\textbf{Type} \to \textbf{Type} : \textbf{Kind}$$

$$\top \to \top : \textbf{Prop}$$

Universes

$$\frac{}{\vdash U_i : U_{i+1}} \qquad \frac{\Gamma \vdash A : U_i \qquad \Gamma, x : A \vdash B : U_i}{\Gamma \vdash (x : A) \to B : U_i}$$

$$nat : \textbf{Type}$$

$$nat \to nat : \textbf{Type}$$

$$\textbf{Type} \to \textbf{Type} : \textbf{Kind}$$

$$\top \to \top : \textbf{Prop}$$

$$(x : \textbf{Type}) \to \top \text{ ✗}$$

$$nat \to \textbf{Type} \text{ ✗}$$

Universes

$$\frac{}{\vdash U_i : U_{i+1}} \qquad \frac{\Gamma \vdash A : U_i \qquad \Gamma, x : A \vdash B : U_j}{\Gamma \vdash (x : A) \to B : U_{rule(i,j)}}$$

Universes

$$\frac{}{\vdash U_i : U_{i+1}} \qquad \frac{\Gamma \vdash A : U_i \qquad \Gamma, x : A \vdash B : U_j}{\Gamma \vdash (x : A) \to B : U_{rule(i,j)}}$$

$$rule(i, 0) :\equiv 0 \text{ (impredicativity)}$$

$$rule(i, j+1) :\equiv max(i, j+1)$$

Universes

$$\frac{}{\vdash U_i : U_{i+1}} \qquad \frac{\Gamma \vdash A : U_i \qquad \Gamma, x : A \vdash B : U_j}{\Gamma \vdash (x : A) \to B : U_{rule(i,j)}}$$

$$\begin{aligned} rule(i, 0) &:\equiv 0 \text{ (impredicativity)} \\ rule(i, j+1) &:\equiv max(i, j+1) \end{aligned}$$

$$(x : \textbf{Type}) \to \top : \textbf{Prop}$$

$$(X : \textbf{Type}) \to X \to X \to \textbf{Prop} : \textbf{Kind}$$

Universes

$$\frac{}{\vdash U_i : U_{i+1}} \qquad \frac{\Gamma \vdash A : U_i \qquad \Gamma, x : A \vdash B : U_j}{\Gamma \vdash (x : A) \to B : U_{rule(i,j)}}$$

$$rule(i, 0) :\equiv 0 \text{ (impredicativity)}$$
$$rule(i, j+1) :\equiv max(i, j+1)$$

$$(x : \textbf{Type}) \to \top : \textbf{Prop}$$
$$(X : \textbf{Type}) \to X \to X \to \textbf{Prop} : \textbf{Kind}$$

$$((x : \textbf{Type}) \to x =_{\textbf{Type}} x)\ \top\ ✗$$

Universes

$$\frac{}{\vdash U_i : U_{i+1}} \qquad \frac{\Gamma \vdash A : U_i \qquad i \leq j}{\Gamma \vdash A : U_j}$$

$$\frac{\Gamma \vdash A : U_i \qquad \Gamma, x : A \vdash B : U_j}{\Gamma \vdash (x : A) \to B : U_{rule\ i\ j}}$$

Universes

$$\frac{}{\vdash U_i : U_{i+1}} \qquad \frac{\Gamma \vdash A : U_i}{\Gamma \vdash \uparrow_i^j A : U_{max\ i\ j}}$$

$$\frac{\Gamma \vdash A : U_i \qquad \Gamma, x : A \vdash B : U_j}{\Gamma \vdash (x : A) \to B : U_{rule\ i\ j}}$$

Universes

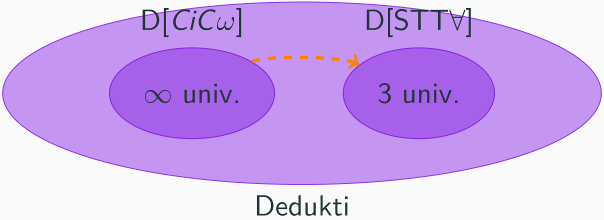
$$\frac{}{\vdash U_i : U_{i+1}} \qquad \frac{\Gamma \vdash A : U_i}{\Gamma \vdash \uparrow_i^j A : U_{max\ i\ j}}$$

$$\frac{\Gamma \vdash A : U_i \qquad \Gamma, x : A \vdash B : U_j}{\Gamma \vdash (x : A) \to B : U_{rule\ i\ j}}$$

$$((x : \textbf{Type}) \to x = x)\ \uparrow_{\textbf{Prop}}^{\textbf{Type}} \top\ ✓$$

$$((x : \textbf{Type}) \to x = x)\ nat\ ✓$$

A minimization problem

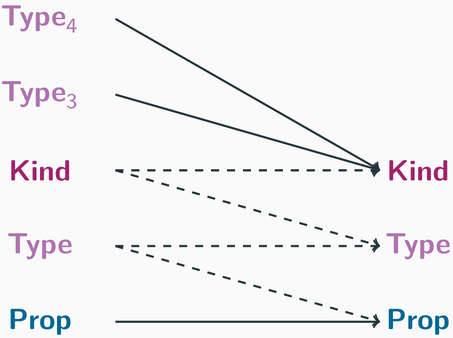


$\mathbf{Type}_4$

$\mathbf{Type}_3$

Kind	**Kind**
Type	**Type**
Prop	**Prop**

$$
\begin{tikzcd}
\mathbf{Type}_4 \arrow[ddr] & \\
\mathbf{Type}_3 \arrow[dr] & \\
\mathbf{Kind} \arrow[r] & \mathbf{Kind} \\
\mathbf{Type} \arrow[r] & \mathbf{Type} \\
\mathbf{Prop} \arrow[r] & \mathbf{Prop}
\end{tikzcd}
$$

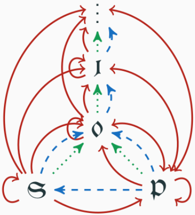


$$\frac{}{\vdash U_i : U_{i+1}}$$

Universo's algorithm

1. Elaboration: Replace every universe by a fresh variable
2. Checking: Generate constraints by type checking the terms (with an implementation of Dedukti)
3. Resolution: Solve the constraints (using an SMT solver)
4. Reconstruction: Replace the solution found for every terms

Coq as a CTS



Demo

Let's use Universo to see whether the proofs using the STT$\forall$ representation can translated into the Coq representation with 3 universes!

Conclusion

Question

What are the advantages of using Dedukti for interoperability?

Dedukti's advantages

- Dedukti aims to be a standard to write logics
- Implementing this standard or relevant part of this standard is rather easy
- (Higher-Order) rewriting is a powerful mechanism to embed logics (encodings are small) and to transform Dedukti terms
- Dedukti's encodings highlight common features of several logics
- Dedukti's encodings allow to better understand the object logic (both practically and theoretically)

Features in Dedukti

Given a feature of a logic (inductive types, universes, classical connectives, eta-reduction, ...):

- Their encoding does not depend on the object logic (empirical fact)
- There might exist several variants in Dedukti (which is a good property)

Main takeaway

Scalability of proof interoperability of proofs with Dedukti depends on the ability to encode features separately and to combine them.