Encoding Logics in the Logical Framework using Rewriting

Gaspard Férey

ISR Short Talks

Paris, July 1, 2019

Logical Framework you say?

A type system

- ✗ No polymorphism
  Types cannot depend on other types: $\forall\ \mathtt{A} : \mathbb{P}\ ,\ \mathtt{A} \to \mathtt{A}$ ✗
- ✗ Only two sorts
  Objects inhabit Types and are never Types themselves
- ✓ Dependent types: $\forall\ \mathtt{x} : \mathbb{N}\ ,\ \mathtt{x} = \mathtt{x}$
- ✓ Rewriting: `rule x + (S y) --> S (x + y)`

Still great to encode logics!

Let's encode logic!

Coq

```
Parameter A : ℙ .
Theorem my_proof : A → A.
Proof.
  exact ( λ (x:A) ⇒ x ).
Qed.
```

Dedukti

Let's encode logic!

Coq

```
Parameter A : ℙ .
Theorem my_proof : A → A.
Proof.
  exact ( λ (x:A) ⇒ x ).
Qed.
```

Dedukti

```
A : Type.
```

Let's encode logic!

Coq

```
Parameter A : ℙ .
Theorem my_proof : A → A.
Proof.
  exact ( λ (x:A) ⇒ x ).
Qed.
```

Dedukti

```
A : Type.
def my_proof : A -> A.
```

Let's encode logic!

Coq

```
Parameter A : ℙ .
Theorem my_proof : A → A.
Proof.
  exact ( λ (x:A) ⇒ x ).
Qed.
```

Dedukti

```
A : Type.
def my_proof : A -> A.
rule my_proof --> x => x.
```

Let's encode logic!

Coq

```
Parameter A : ℙ .
Theorem my_proof : A → A.
Proof.
  exact ( λ (x:A) ⇒ x ).
Qed.
```

Dedukti

```
A : Type.
def my_proof : A -> A.
rule my_proof --> x => x.
```

✔ Typechecks

Let's encode logic!

Coq

```
Theorem my_proof : ∀ A:ℙ , A → A.
Proof.
  exact (λ  (A:ℙ ) ⇒ λ (x:A) ⇒ x).
Qed.
```

Dedukti

Let's encode logic!

Coq

```
Theorem my_proof : ∀ A:ℙ , A → A.
Proof.
  exact (λ  (A:ℙ ) ⇒ λ (x:A) ⇒ x).
Qed.
```

Dedukti

```
def my_proof : (A:Type) -> A -> A.
```

Let's encode logic!

Coq

```
Theorem my_proof : ∀ A:ℙ , A → A.
Proof.
  exact (λ  (A:ℙ ) ⇒ λ (x:A) ⇒ x).
Qed.
```

Dedukti

```
def my_proof : (A:Type) -> A -> A.
```

✗ Fails because no polymorphism

Let's encode logic!

Coq

```
Theorem my_proof : ∀ A:ℙ , A → A.
Proof.
  exact (λ (A:ℙ ) ⇒ λ (x:A) ⇒ x).
Qed.
```

Dedukti

```
Prop : Type.
def my_proof : (A:Prop) -> A -> A.
```

Let's encode logic!

Coq

```
Theorem my_proof : ∀ A:ℙ , A → A.
Proof.
  exact (λ  (A:ℙ ) ⇒ λ (x:A) ⇒ x).
Qed.
```

Dedukti

```
Prop : Type.
def my_proof : (A:Prop) -> A -> A.
```

✗ Fails because `A` is no longer a `Type`

Let's encode logic!

Coq

```
Theorem my_proof : ∀ A:ℙ , A → A.
Proof.
  exact (λ (A:ℙ ) ⇒ λ (x:A) ⇒ x).
Qed.
```

Dedukti

```
Prop : Type.
Proof : Prop -> Type.

def my_proof : (A:Prop) -> Proof A -> Proof A.
rule my_proof --> A => x => x.
```

Let's encode logic!

Coq

```
Theorem my_proof : ∀ A:ℙ , A → A.
Proof.
  exact (λ (A:ℙ ) ⇒ λ (x:A) ⇒ x).
Qed.
```

Dedukti

```
Prop : Type.
Proof : Prop -> Type.

def my_proof : (A:Prop) -> Proof A -> Proof A.
rule my_proof --> A => x => x.
```

✔ Typechecks

Let's encode logic!

Coq

```
Definition True : ℙ := ∀ A:ℙ , A → A.
Theorem my_proof : True.
Proof.
  exact (λ (A:ℙ ) ⇒ λ (x:A) ⇒ x).
Qed.
```

Dedukti

```
Prop : Type.
Proof : Prop -> Type.

def True : Prop.
rule True --> ???.

def my_proof : Proof True.
rule my_proof --> A => x => x.
```

Let's encode logic!

Coq

```
Definition True : ℙ := ∀ A:ℙ , A → A.
Theorem my_proof : True.
Proof.
  exact (λ  (A:ℙ ) ⇒ λ (x:A) ⇒ x).
Qed.
```

Dedukti

```
Prop : Type.
Proof : Prop -> Type.

def True : Prop.
rule True --> ???.

def my_proof : Proof True.
rule my_proof --> A => x => x.
```

✗ No "∀ A:ℙ , A → A" object

Let's encode logic!

Coq

```
Definition True : ℙ := ∀ A:ℙ , A → A.
Theorem my_proof : True.
Proof.
  exact (λ  (A:ℙ ) ⇒ λ (x:A) ⇒ x).
Qed.
```

Dedukti

```
Prop : Type.
Proof : Prop -> Type.

implies : Prop -> Prop -> Prop.
(; Builds "A -> B"   from A and B. ;)

forall : (Prop -> Prop) -> Prop.
(; Builds "forall A, P(A)" from P  ;)

def True : Prop.
rule True --> forall (A => (implies A A)).
```

Let's encode logic!

Coq

```
Definition True : ℙ := ∀ A:ℙ , A → A.
Theorem my_proof : True.
Proof.
  exact (λ  (A:ℙ ) ⇒ λ (x:A) ⇒ x).
Qed.
```

Dedukti

```
Prop : Type.
Proof : Prop -> Type.

implies : Prop -> Prop -> Prop.
(; Builds "A -> B"   from A and B. ;)

forall : (Prop -> Prop) -> Prop.
(; Builds "forall A, P(A)" from P  ;)

def True : Prop.
rule True --> forall (A => (implies A A)).
```

✔ Typechecks so far

Let's encode logic!

Coq

```
Definition True : ℙ := ∀ A:ℙ , A → A.
Theorem my_proof : True.
Proof.
  exact (λ  (A:ℙ ) ⇒ λ (x:A) ⇒ x).
Qed.
```

Dedukti

```
Prop : Type.
implies : Prop -> Prop -> Prop.
forall : (Prop -> Prop) -> Prop.
def True : Prop := forall (A => (implies A A)).

Proof : Prop -> Type.



def my_proof : Proof True.
rule my_proof --> A => x => x.
```

Let's encode logic!

Coq

```
Definition True : ℙ := ∀ A:ℙ , A → A.
Theorem my_proof : True.
Proof.
  exact (λ  (A:ℙ ) ⇒ λ (x:A) ⇒ x).
Qed.
```

Dedukti

```
Prop : Type.
implies : Prop -> Prop -> Prop.
forall : (Prop -> Prop) -> Prop.
def True : Prop := forall (A => (implies A A)).

Proof : Prop -> Type.



def my_proof : Proof True.
rule my_proof --> A => x => x.
```

✗ Wrong type

Let's encode logic!

Coq

```
Definition True : ℙ := ∀ A:ℙ , A → A.
Theorem my_proof : True.
Proof.
  exact (λ  (A:ℙ ) ⇒ λ (x:A) ⇒ x).
Qed.
```

Dedukti

```
Prop : Type.
implies : Prop -> Prop -> Prop.
forall : (Prop -> Prop) -> Prop.
def True : Prop := forall (A => (implies A A)).

def Proof : Prop -> Type.
rule Proof (implies a b)  -->  Proof a -> Proof b.
rule Proof (forall P)     -->  (A:Prop) -> Proof (P A).

def my_proof : Proof True.
rule my_proof --> A => x => x.
```

Let's encode logic!

Coq

```
Definition True : ℙ := ∀ A:ℙ , A → A.
Theorem my_proof : True.
Proof.
  exact (λ  (A:ℙ ) ⇒ λ (x:A) ⇒ x).
Qed.
```

Dedukti

```
Prop : Type.
implies : Prop -> Prop -> Prop.
forall : (Prop -> Prop) -> Prop.
def True : Prop := forall (A => (implies A A)).

def Proof : Prop -> Type.
rule Proof (implies a b)  -->  Proof a -> Proof b.
rule Proof (forall P)     -->  (A:Prop) -> Proof (P A).

def my_proof : Proof True.
rule my_proof --> A => x => x.
```

✔ Typechecks

Let's encode logic!

Coq

```
Definition True : ℙ := ∀ A:ℙ , A → A.
Theorem my_proof : True.
Proof.
  exact (λ (A:ℙ ) ⇒ λ (x:A) ⇒ x).
Qed.
```

Dedukti

```
Prop : Type.
implies : Prop -> Prop -> Prop.
forall : (Prop -> Prop) -> Prop.
def True : Prop := forall (A => (implies A A)).

def Proof : Prop -> Type.
rule Proof (implies a b)  -->  Proof a -> Proof b.
rule Proof (forall P)     -->  (A:Prop) -> Proof (P A).

def my_proof : Proof True.
rule my_proof --> A => x => x.
```

✔ `Proof True` $\hookrightarrow^*$ `(A:Prop) -> Proof a -> Proof a`

What more could I possibly want?

The rest of Coq

- Calculus of Constructions
- Infinite hierarchy of universes
- Subtyping
- Inductive constructions
- Universe polymorphism

Some theory

- Confluent, terminating, consistent encoding
- Correct, conservative translation function
- Implementation working on a significant proof dataset

Thanks!

Now let's have a drink!