Deduction modulo rewriting II

In the early ages

Each time we wrote a piece of software (for example: text processing system) we chose a representation for the data

- code “a” with 97 and “b” with 98
- or the other way around

Involuntarily defined a format

In modern times

We define a format first
Software has to comply to the format

ASCII, TCP / IP, HTTP, HTML, UNICODE...

But not yet in the realm of formal proofs

"A Coq proof of the four color theorem"

Problems: interoperability, sustainability

I. From interoperability to reverse mathematics

Why is it more difficult with formal proofs?

Because we cannot go too far

Euclidean geometry $\not\leftrightarrow$ Hyperbolic geometry

$\mathrm{ZF} \not\leftrightarrow \mathrm{ZFC}$

But...

A proof in ZF can be "translated" to ZFC

A proof in ZFC that does not use the axiom of choice can be "translated" to ZF

Proof transformation

There exists a basis of $\mathbb{R}^2$

- by the incomplete basis theorem (axiom of choice)
- $\langle 1, 0 \rangle$, $\langle 0, 1 \rangle$

automatically (for example: constructivization) or by hand

Reverse mathematics as the basis of interoperability

Reformulating the project of reverse mathematics

- Formal proofs, not pencil-paper-LaTeXones
- Expressive theories (Set theory, Type theory...) and not fragments of arithmetic
- Analyze proofs before (possibly) transforming them

II. Logical Frameworks

The interoperability ZF / ZFC possible because ZF and ZFC expressed in the same logical framework: predicate logic

In predicate logic, a theory: several axioms

Permits to raise the question: which axioms are used in a proof π

The revolution of predicate logic

Since Euclid: geometry, arithmetic, set theory... each system its syntax, its notion of proof...

Hilbert and Ackermann (1928): a common predicate logic

A common framework for geometry, arithmetic, set theory...
Sharing connectives, deduction rules...

A theory: symbols and axioms

But a short revolution

Predicate logic: simplification of Russell's Type theory (*Principia Mathematica*)
But no reformulation of Russell's Type theory in predicate logic

Soon (1940) Church: a new formulation of Type theory (based on λ-calculus) impossible to express in predicate logic (λ binds)

1970, 1985... Martin-Löf's type theory, the Calculus of constructions... not in predicate logic

Three attitudes

- Consider logical framework as a dead concept
- Express Russell's type theory, Church's, Martin-Löf's, the Calculus of constructions... in predicate logic by will of by force
- Extend predicate logic to a better logical framework

The limits of predicate logic

- No bound variables ($\lambda x\ x$)
- No syntax for proofs
- No notion of computation
- No good notion of cut
- Classical and not constructive

New logical frameworks

- No bound variables ($\lambda x\ x$): λ-Prolog, Isabelle, $\lambda\Pi$-calculus
- No syntax for proofs: $\lambda\Pi$-calculus
- No notion of computation: Deduction modulo theory
- No good notion of cut: Deduction modulo theory
- Classical and not constructive: Ecumenical logic

The $\lambda\Pi$-calculus modulo theory that generalizes them all

What is the $\lambda\Pi$-calculus modulo theory?

- A lambda-calculus: same as predicate logic: $0 : \mathit{nat}$,
  $S : \mathit{nat} \to \mathit{nat}$, $\mathit{even} : \mathit{nat} \to o$, $\mathit{even}(S(0)) : o$
  But also binding symbols $\mathit{let}\ x = 4\ \mathit{in}\ x + 1$ expressed as
  $\mathit{let}\ 4\ \lambda x\ x + 1$
  $\mathit{let} : \mathit{nat} \to (\mathit{nat} \to \mathit{nat}) \to \mathit{nat}$
- Dependent types: to express proofs as terms
- Rewrite rules: to express the theories

Dedukti

An implementation of the $\lambda\Pi$-calculus modulo theory

Just a proof-checker
re-implementable in two weeks

Defining a theory in Dedukti

No universal method
But several paradigmatic examples

- Any (finite) theory expressed in Predicate logic
- Axiom schemes
- Simple type theory (without and with polymorphism)
- Pure type systems (CoC...)
- Inductive types
- Universes

III. Simple type theory in Dedukti

Simple types

- *nat* and *o* are simple types,
- if A and B are simple types, then $A \to B$ is a simple type.

In predicate logic: a many-sorted theory

Language

An infinite number of variables of each type,
constants: $\Rightarrow$ of type $o \to o \to o$, $\forall_A$ of type $(A \to o) \to o$, $=$ ...

Simply typed lambda-calculus

- variables and constants: if x variable or constant of type A then $x : A$
- abstractions: if x variable of type A and $t : B$, then $\lambda x : A\ t : A \to B$
- applications: if $t : A \to B$ and $u : A$ then $(t\ u) : B$,

o type of propositions

In predicate logic: for application, for each pair A, B, $\alpha_{A,B}$ of function symbol arity $\langle A \to B, A, B \rangle$
But λ binds: replace lambda-calculus with combinators...
Propositions are not terms: a predicate symbol ε or ariy $\langle o \rangle$

Examples

Types:

$nat \to nat$

Terms:

$\lambda x : nat\ x$

Propositions:

$\lambda x : nat\ x = \lambda x : nat\ x$

$\forall_o\ \lambda X : o\ (X \Rightarrow X)$

Proofs

$$\dfrac{\dfrac{\overline{X \vdash X}}{\vdash X \Rightarrow X}}{\vdash \forall_o\ \lambda X : o\ (X \Rightarrow\ X)}$$

Simple type theory in DEDUKTI

$$
\begin{array}{rcl}
\mathit{type} & : & \mathit{Type} \\
\eta & : & \mathit{type} \to \mathit{Type} \\
o & : & \mathit{type} \\
\mathit{nat} & : & \mathit{type} \\
\mathit{arrow} & : & \mathit{type} \to \mathit{type} \to \mathit{type} \\
\varepsilon & : & (\eta\ o) \to \mathit{Type} \\
\Rightarrow & : & (\eta\ o) \to (\eta\ o) \to (\eta\ o) \\
\forall & : & \Pi x : \mathit{type}\ (((\eta\ x) \to (\eta\ o)) \to (\eta\ o))
\end{array}
$$

$$
\begin{array}{rcl}
(\eta\ (\mathit{arrow}\ x\ y)) & \longrightarrow & (\eta\ x) \to (\eta\ y) \\
(\varepsilon\ (\Rightarrow\ x\ y)) & \longrightarrow & (\varepsilon\ x) \to (\varepsilon\ y) \\
(\varepsilon\ (\forall\ x\ y)) & \longrightarrow & \Pi z : (\eta\ x)\ (\varepsilon\ (y\ z))
\end{array}
$$

Examples

Types: $\mathit{nat} \to \mathit{nat}$ expressed as $(\mathit{arrow}\ \mathit{nat}\ \mathit{nat})$ of type type
Then to $(\eta\ (\mathit{arrow}\ \mathit{nat}\ \mathit{nat}))$ of type Type that reduces to
$(\eta\ \mathit{nat}) \to (\eta\ \mathit{nat})$

Terms: $\lambda x : \mathit{nat}\ x$ expressed as $\lambda x : (\eta\ \mathit{nat})\ x$ of type
$(\eta\ \mathit{nat}) \to (\eta\ \mathit{nat})$

Propositions: $\forall_o\ \lambda X : o\ (X \Rightarrow X)$ expressed as
$\forall\ o\ \lambda X : (\eta\ o)\ (\Rightarrow\ X\ X)$ of type $(\eta\ o)$
Then to $(\varepsilon\ (\forall\ o\ \lambda X : (\eta\ o)\ (\Rightarrow\ X\ X)))$ of type Type that reduces to $\Pi X : (\eta\ o)\ ((\varepsilon\ X) \to (\varepsilon\ X))$.

Proofs: well-know expressed as $\lambda X : (\eta\ o)\ \lambda \alpha : (\varepsilon\ X)\ \alpha$ of type
$\Pi X : (\eta\ o)\ ((\varepsilon\ X) \to (\varepsilon\ X))$

IV. Reverse mathematics in Dedukti

Simple type theory in DEDUKTI

$$
\begin{array}{rcl}
\mathit{type} & : & \mathit{Type} \\
\eta & : & \mathit{type} \to \mathit{Type} \\
o & : & \mathit{type} \\
\mathit{nat} & : & \mathit{type} \\
\mathit{arrow} & : & \mathit{type} \to \mathit{type} \to \mathit{type} \\
\varepsilon & : & (\eta\ o) \to \mathit{Type} \\
\Rightarrow & : & (\eta\ o) \to (\eta\ o) \to (\eta\ o) \\
\forall & : & \Pi x : \mathit{type}\ (((\eta\ x) \to (\eta\ o)) \to (\eta\ o))
\end{array}
$$

$$
\begin{array}{rcl}
(\eta\ (\mathit{arrow}\ x\ y)) & \longrightarrow & (\eta\ x) \to (\eta\ y) \\
(\varepsilon\ (\Rightarrow\ x\ y)) & \longrightarrow & (\varepsilon\ x) \to (\varepsilon\ y) \\
(\varepsilon\ (\forall\ x\ y)) & \longrightarrow & \Pi z : (\eta\ x)\ (\varepsilon\ (y\ z))
\end{array}
$$

The Calculus of constructions in DEDUKTI

$$
\begin{array}{rcl}
\mathit{type} & : & \mathit{Type} \\
\eta & : & \mathit{type} \to \mathit{Type} \\
o & : & \mathit{type} \\
\mathit{nat} & : & \mathit{type} \\
\mathit{arrow} & : & \Pi x : \mathit{type}\ ((\eta\ x) \to \mathit{type}) \to \mathit{type}) \\
\varepsilon & : & (\eta\ o) \to \mathit{Type} \\
\Rightarrow & : & \Pi x : (\eta\ o)\ ((\varepsilon\ x) \to (\eta\ o)) \to (\eta\ o)) \\
\forall & : & \Pi x : \mathit{type}\ ((\eta\ x) \to (\eta\ o)) \to (\eta\ o)) \\
\pi & : & \Pi x : (\eta\ o)\ ((\varepsilon\ x) \to \mathit{type}) \to \mathit{type})
\end{array}
$$

$$
\begin{array}{rcl}
(\eta\ (\mathit{arrow}\ x\ y)) & \longrightarrow & \Pi z : (\eta\ x)\ (\eta\ (y\ z)) \\
(\varepsilon\ (\Rightarrow\ x\ y)) & \longrightarrow & \Pi z : (\varepsilon\ x)\ (\varepsilon\ (y\ z)) \\
(\varepsilon\ (\forall\ x\ y)) & \longrightarrow & \Pi z : (\eta\ x)\ (\varepsilon\ (y\ z)) \\
(\eta\ (\pi\ x\ y)) & \longrightarrow & \Pi z : (\varepsilon\ x)\ (\eta\ (y\ z))
\end{array}
$$

A comparison

- *arrow* dependent in the Calculus of constructions but not in Simple type theory
- Same for $\Rightarrow$
- An extra symbol π in the Calculus of constructions: express functions mapping proofs to terms

Reverse mathematics in Dedukti

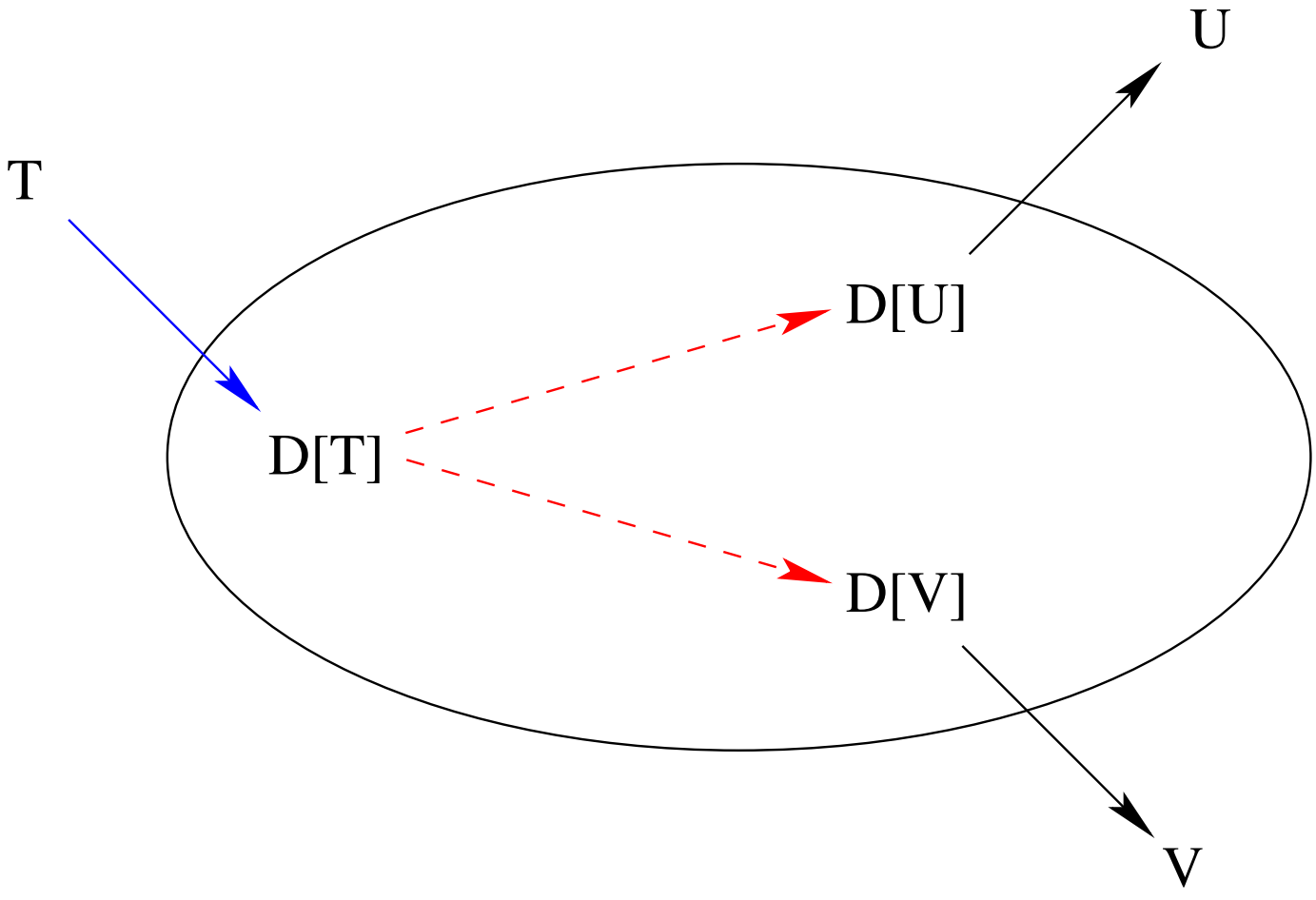
- All proofs in Simple type theory can be translated to the Calculus of constructions
- The proofs in the Calculus of constructions that do not use these three features can be translated to Simple type theory

(not the others: genuine Calculus of constructions proofs)

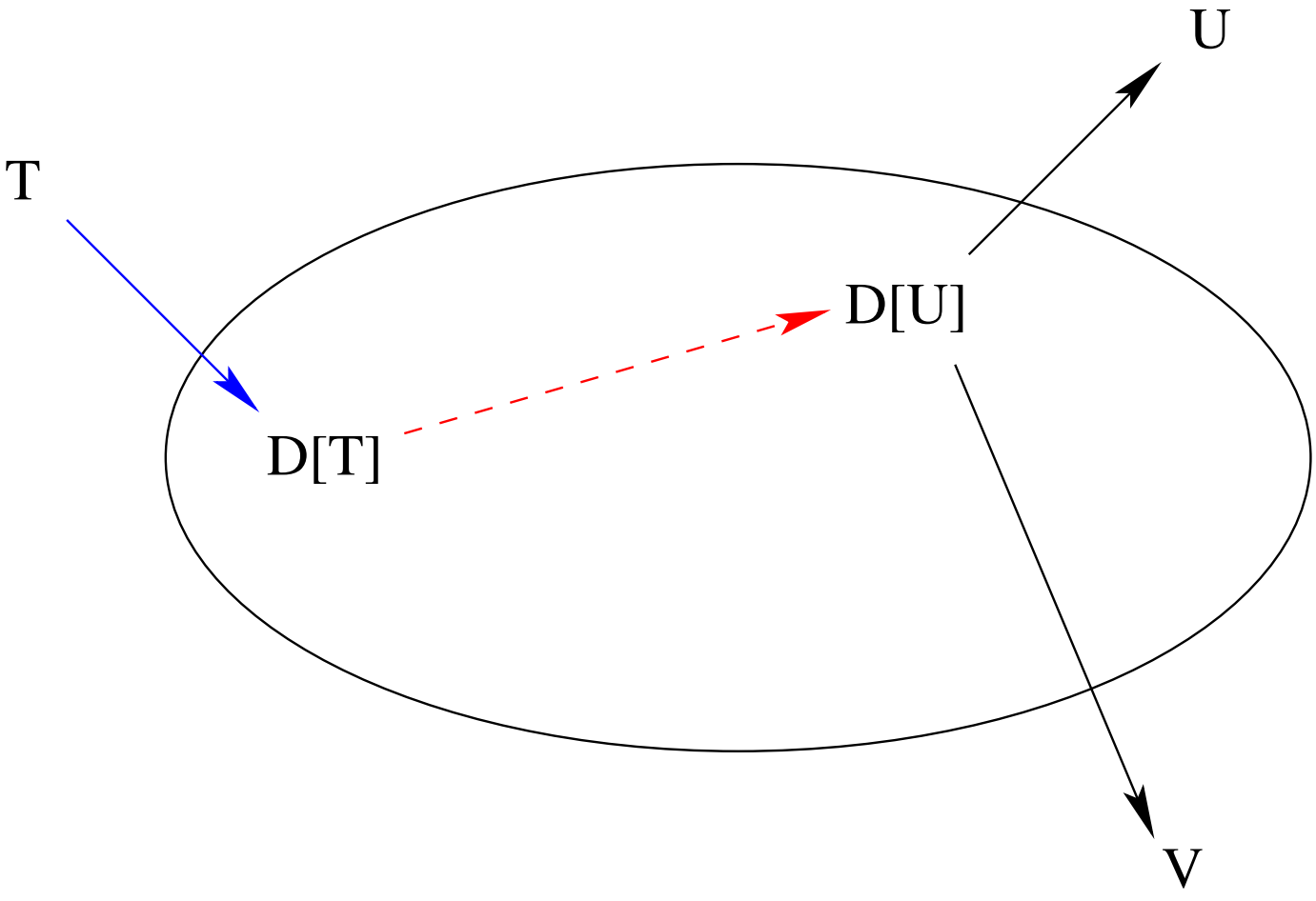
For example: all the proofs of the arithmetic library of Matita

"First" proof of Fermat's little theorem in constructive Simple type theory

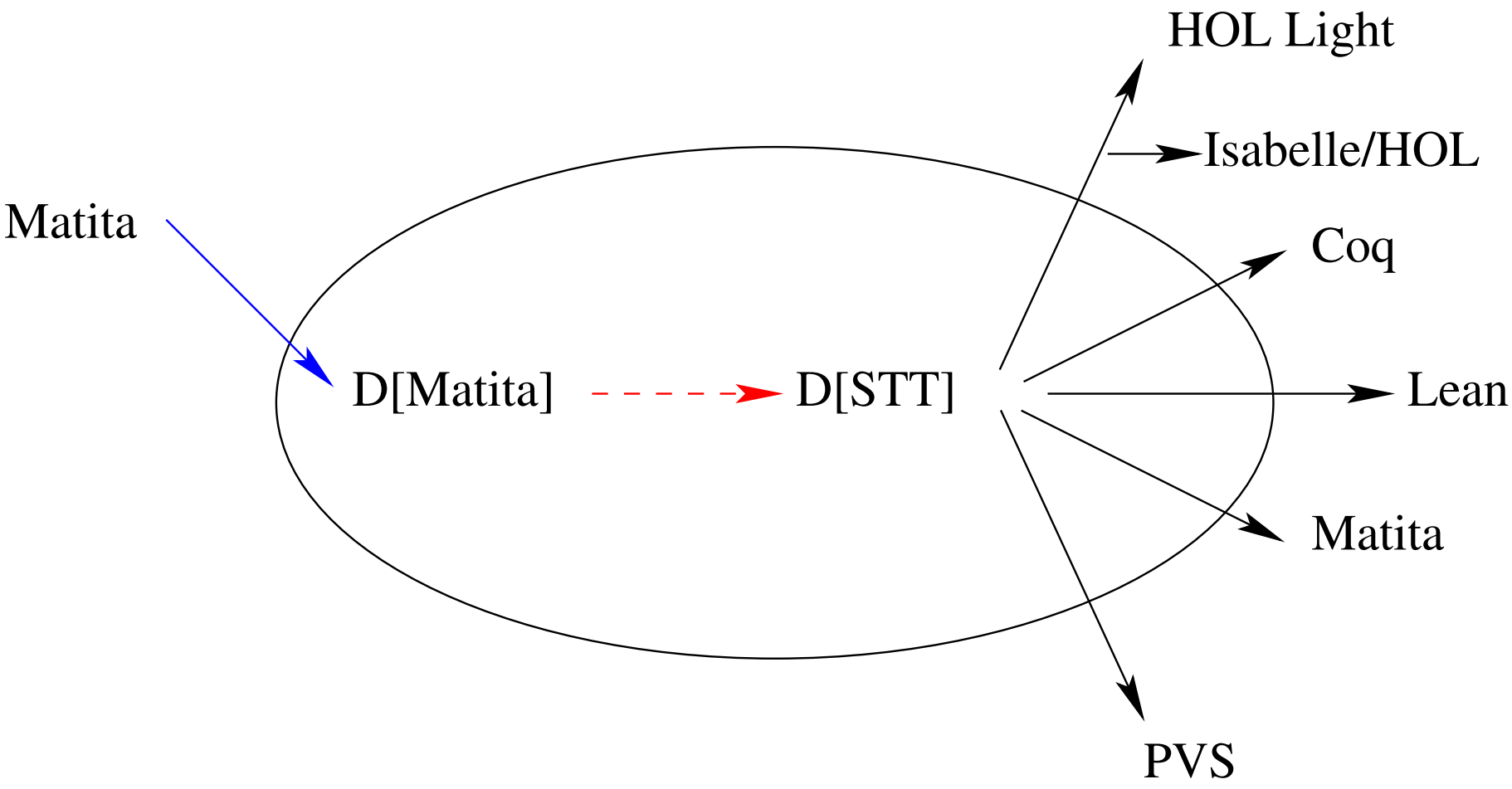
Proof translation



But also



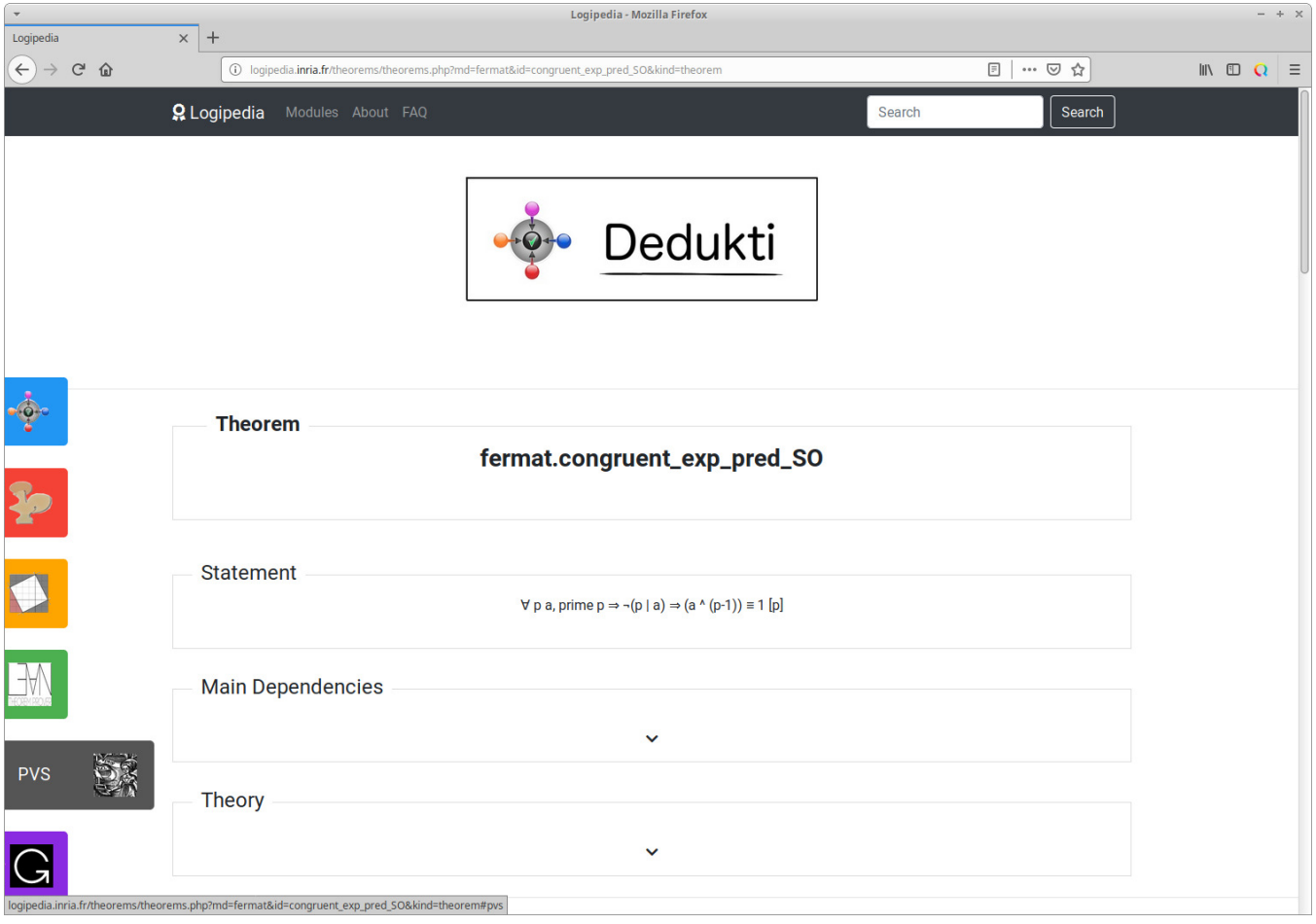
An example


HOL Light
Isabelle/HOL
Matita
Coq
D[Matita]
D[STT]
Lean
Matita
PVS

Collecting all the proofs in a single data base

LOGIPEDIA: an encyclopedia of proofs expressed

- in various theories
- in DEDUKTI



http://logipedia.science

Why does it work?

Because proof systems implement very expressive theories and use only a tiny part of it

Two early empirical evidences

- Proof systems: very different theories, but very similar libraries
- Mathematicians are not very interested in the axioms used in their proofs: any theory seems to fit