Logipedia

How to use it, and how to contribute to it
http://logipedia.science
Logical Frameworks

A logical system (Euclidean geometry, set theory, Simple type theory, the Calculus of constructions...) should not be defined as independent system

They should be expressed in a Logical framework

Logical Frameworks: Predicate logic (1928), \( \lambda \)-Prolog, Isabelle, Pure type systems, the \( \lambda \Pi \)-calculus (LF), Deduction modulo theory, the \( \lambda \Pi \)-calculus modulo theory (Dedukti)

Each theory breaks down into a number of axioms / rewrite rules

Permits to analyze which proof uses which axiom / rewrite rule (reverse mathematics)
An encyclopedia of proofs expressed

- in various theories
- in **Dedukti**
Proof translation
But also

\[ D[U] \]

\[ D[T] \]

\[ U \]

\[ V \]
An example

Matita
D[CIC] -> D[STTfa]

OpenTheory
Coq
Lean
Matita
PVS
I. Defining a theory in Dedukti
No universal method

Depends on the theory
But several “paradigmatic” examples in *Dedukti: a Logical Framework based on the lambda-Pi-Calculus Modulo Theory*.  

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

- Inductive types
- Universes (with universe polymorphism)
- Proof irrelevance
- Predicate subtyping
An example: Simple type theory

\[
\begin{align*}
type & : \quad Type \\
Te & : \quad type \rightarrow Type \\
o & : \quad type \\
nat & : \quad type \\
arrow & : \quad type \rightarrow type \rightarrow type \\
Pf & : \quad (Te o) \rightarrow Type \\
\Rightarrow & : \quad (Te o) \rightarrow (Te o) \rightarrow (Te o) \\
\forall & : \quad \Pi a : type (((Te a) \rightarrow (Te o)) \rightarrow (Te o)) \\
\end{align*}
\]

\[
\begin{align*}
(Te (arrow x y)) & \quad \rightarrow \quad (Te x) \rightarrow (Te y) \\
(Pf (\Rightarrow x y)) & \quad \rightarrow \quad (Pf x) \rightarrow (Pf y) \\
(Pf (\forall x y)) & \quad \rightarrow \quad \Pi z : (Te x) (Pf (y z))
\end{align*}
\]
Examples

Types: \( \text{nat} \rightarrow \text{nat} \) expressed as \((\text{arrow nat nat})\) of type type
Then to \((\text{Te} (\text{arrow nat nat}))\) of type \text{Type} that reduces to \((\text{Te nat}) \rightarrow (\text{Te nat})\)

Terms: \(\lambda x : \text{nat} \; x\) expressed as \(\lambda x : (\text{Te nat}) \; x\) of type \((\text{Te nat}) \rightarrow (\text{Te nat})\)

Propositions: \(\forall X : o \; (X \Rightarrow X)\) expressed as
\(\forall o \; \lambda X : (\text{Te o}) \; (\Rightarrow \; X \; X)\) of type \((\text{Te o})\)
Then to \((\text{Pf} \; (\forall o \; \lambda X : (\text{Te o}) \; (\Rightarrow \; X \; X)))\) of type \text{Type} that reduces to \(\Pi X : (\text{Te o}) \; ((\text{Pf} \; X) \rightarrow (\text{Pf} \; X))\).

Proofs: well-known expressed as \(\lambda X : (\text{Te o}) \; \lambda \alpha : (\text{Pf} \; X) \; \alpha\) of type \(\Pi X : (\text{Te o}) \; ((\text{Pf} \; X) \rightarrow (\text{Pf} \; X))\)
II. Exporting proofs from Dedukti
Three types of systems

- Those with explicit proof terms (Automath-like: Coq, Matita, Lean, Agda...)
- Those with predictable tactics (LCF-like: HOL Light, Isabelle/HOL...)
- Those with neither (PVS-like: PVS)
Three types of systems

- Those with explicit proof terms (Automath-like: Coq, Matita, Lean, Agda...)
  Just translate the proof term
- Those with predictable tactics (LCF-like: HOL Light, Isabelle/HOL...)
  Generate tactics (at the level of Natural deduction rules)
- Those with neither (PVS-like: PVS)
  A tree such that a proposition labeling a node is not too difficult to prove from those labeling its children and cut
  Example: \( a = b, \ b = a \ldots \)
  State \( \vdash a = b \)
  Cut on \( b = a \)
  Prove automatically \( b = a \vdash a = b \)
  Continue with \( \vdash b = a \)
Easy to do

One day, one week... depending on the system
III. Importing proofs to Dedukti
More difficult

Usually requires to instrument the source system

But done with Matita, HOL Light, FoCaLiZe, iProver, Zenon, ArchSAT

- Zenon and ArchSAT have been designed with a Dedukti output
- HOL Light has a output to some proof certificates OpenTheory, that we could translate to Dedukti
Same three types of systems

- Those with explicit proof terms (Automath-like)
  Just translate the proof term
- Those with a small set of primitive tactics (LCF-like) used to build the others
  Instrument the primitive tactics only
- Those with neither (PVS-like), in particular iProver Ford technique (again)
  Output a list of intermediate steps, use an automated theorem prover (that output Dedukti proofs) to fill the gaps, rebuild the puzzle from the pieces
IV. Reverse mathematics in Dedukti
(A slight extension of) the Calculus of constructions as a theory in in Dedukti

\[
\begin{align*}
type & : \quad \text{Type} \\
Te & : \quad type \rightarrow \text{Type} \\
o & : \quad type \\
nat & : \quad type \\
arrow & : \quad \Pi x : type ((Te x) \rightarrow type) \rightarrow type \\
Pf & : \quad (Te o) \rightarrow \text{Type} \\
\Rightarrow & : \quad \Pi x : (Te o) (((Pf x) \rightarrow (Te o)) \rightarrow (Te o)) \\
\forall & : \quad \Pi x : type (((Te x) \rightarrow (Te o)) \rightarrow (Te o)) \\
\pi & : \quad \Pi x : (Te o) (((Pf x) \rightarrow type) \rightarrow type)
\end{align*}
\]

\[
\begin{align*}
(Te (arrow \times y)) & \quad \rightarrow \quad \Pi z : (Te x) (Te (y z)) \\
(Pf (\Rightarrow \times y)) & \quad \rightarrow \quad \Pi z : (Pf x) (Pf (y z)) \\
(Pf (\forall \times y)) & \quad \rightarrow \quad \Pi z : (Te x) (Pf (y z)) \\
(Te (\pi \times y)) & \quad \rightarrow \quad \Pi z : (Pf x) (Te (y z))
\end{align*}
\]
(A slight extension of) the Calculus of constructions as a theory in Dedukti

\[
\begin{align*}
type & : \ Type \\
Te & : \ type \rightarrow \ Type \\
o & : \ type \\
nat & : \ type \\
arrow & : \ \Pi x : \ type \ (((Te \ x) \rightarrow \ type) \rightarrow \ type) \\
Pf & : \ (Te \ o) \rightarrow \ Type \\
\Rightarrow & : \ \Pi x : \ (Te \ o) \ (((Pf \ x) \rightarrow (Te \ o)) \rightarrow (Te \ o)) \\
\forall & : \ \Pi x : \ type \ (((Te \ x) \rightarrow (Te \ o)) \rightarrow (Te \ o)) \\
\pi & : \ \Pi x : \ (Te \ o) \ (((Pf \ x) \rightarrow \ type) \rightarrow \ type) \\
\end{align*}
\]

\[
\begin{align*}
(Te \ (arrow \ x \ y)) & \rightarrow \ \Pi z : \ (Te \ x) \ (Te \ (y \ z)) \\
(Pf \ (\Rightarrow \ x \ y)) & \rightarrow \ \Pi z : \ (Pf \ x) \ (Pf \ (y \ z)) \\
(Pf \ (\forall \ x \ y)) & \rightarrow \ \Pi z : \ (Te \ x) \ (Pf \ (y \ z)) \\
(Te \ (\pi \ x \ y)) & \rightarrow \ \Pi z : \ (Pf \ x) \ (Te \ (y \ z))
\end{align*}
\]
Comparing the theories

arrow in Simple type theory

$$\Pi x : type \ (type \rightarrow type)$$

in the Calculus of constructions

$$\Pi x : type \ (((Te \ x) \rightarrow type) \rightarrow type)$$

In the Calculus of constructions, dependent arrow: in $$A \rightarrow B$$ (written $$\Pi x : A \ B$$), $$B$$ can contain a variable $$x$$ of type $$A$$

Same for $$\Rightarrow$$

($$\forall$$ is dependent is both theories)

An extra constant $$\pi$$ in the Calculus of constructions: typing functions mapping proofs to terms
Analyzing proofs expressed in the Calculus of constructions

A subset $S$ of the proofs expressed in the Calculus of constructions

- do not use the dependency of *arrow*
- do not use the dependency of the symbol $\Rightarrow$,
- do not use the symbol $\pi$

Many proofs expressed in the Calculus of constructions in $S$
Translating proofs to Simple type theory

A proof in the Calculus of constructions

In S
Translation to Simple type theory:
replace \( \text{arrow} \ A \ \lambda x : (\, Te \ A \,) \, B \) with \( \text{arrow} \ A \ B \)
(similar for \( \Rightarrow \))

Not in S
Genuinely uses a feature of the Calculus of constructions that does not exist in Simple type theory
Cannot be expressed in Simple type theory
Same as in ZFC: genuinely uses the axiom of choice: not in ZF
Weaker and weaker

Currently: the “first” proof of Fermat’s little theorem in constructive Simple type theory (no full polymorphism, no dependent types, no universes...)

Further: predicative constructive Simple type theory

Further?: PA, fragments of PA...
V. Towards concept alignment
Connectives and quantifiers

Inductive types / $Q_0$
Should be ignored by the library

Making formal the saying: Cauchy sequences or Dedekind cuts immaterial (isomorphic and only structural statements)

But may be: one classical disjunction and one constructive one (Ecumenical systems)
Further

The induction principle

Justified in different ways in different systems (axiom, consequence of the definition of natural numbers...)

Does not matter as long as it is there
Not the first attempt to build a standard or a library

Why will / might it work this time?

- A better understanding of the theories behind the provers (40 years of research in logic)
- Success stories in point to point translations (Coq / HOL Light)
- A logical framework to express these theories (more abstract view)
- Try to accommodate as many people as possible but not all (theories expressed in Dedukti, e.g. predicate subtyping: research effort)
- Analyzing the proofs (reverse mathematics) before we share them (partial translations)
First discussion before we go deeper

Which proof libraries should we target?

Which similar effort should we build upon?

What should we expect from an encyclopedia?