Proof System Interoperability

Frédéric Blanqui

EuroProofNet

(URLs and purple texts are clickable)

Outline

Historical overview on proof system interoperability

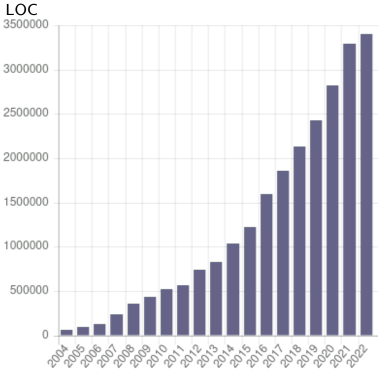
How to encode logics in $\lambda\Pi/\mathcal{R}$?

Example: from HOL-Light to Coq via Lambdapi

Libraries of formal proofs today

Library	Nb files	Nb objects*
Coq Opam	35,000	1,200,000
Isabelle AFP	7,500	280,000
Lean Mathlib	3,200	80,000
Mizar Mathlib	1,400	77,000
HOL-Light Lib	600	35,000
...	...	...

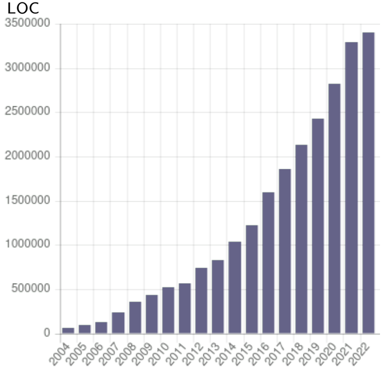
* type, definition, theorem, ...



Libraries of formal proofs today

Library	Nb files	Nb objects*
Coq Opam	35,000	1,200,000
Isabelle AFP	7,500	280,000
Lean Mathlib	3,200	80,000
Mizar Mathlib	1,400	77,000
HOL-Light Lib	600	35,000
...	...	...

* type, definition, theorem, ...



- Every system has its own basic libraries on integers, lists, reals, ...
- Some definitions/theorems are available in one system only and took several man-years to be formalized

Interest of proof system interoperability

- Avoid duplicating developments and losing time
- Facilitate development of new proofs and new systems
- Increase reliability of formal proofs (cross-checking)
- Facilitate validation by certification authorities
- Relativize the choice of a system (school, industry)
- Provide multi-system data to machine learning

Difficulties of proof system interoperability

- Each system is based on different axioms and deduction rules
- It is usually non trivial and sometimes impossible to translate a proof from one system to the other (e.g. a classical proof in an intuitionistic system)

Some milestones

- 1993: QED Manifesto
  DIMACS format for CNF problems
  TPTP format for FOL problems [Sutcliffe & al]
- 1996: HOL90 to NuPRL translator [Howe, statements only]
- 1998: MathML/OpenMath/OMDoc [Kohlhase & al]
- 2003: TPDB format for rewrite systems
  TSTP proof format for ATPs
  SMT-lib format for FOL/T problems
  Flyspeck project with HOL-Light, Coq and Isabelle/HOL
- 2007: Functional PTSs in $\lambda\Pi/\mathcal{R}$ [Cousineau & Dowek]
- 2009: CPF proof format for termination provers
- 2011: Logic Atlas & Integrator [Kohlhase & al]
- 2013: DRAT proof format for SAT solvers [Heule & al]
  MMT/Modules for Mathematical Theories [Rabe & al]
- 2020: Alethe proof format for SMT solvers [Fontaine & al]

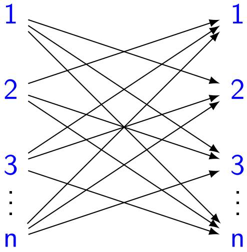
One-to-one translation tools

- HOL90 to NuPRL [Howe 1996, statements only]
- HOL98 to Coq [Denney 2000]
- HOL98 to NuPRL [Naumov et al 2001]

Flyspeck project with HOL-Light, Coq and Isabelle/HOL [2003]

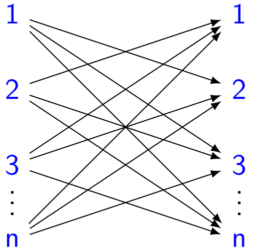
- HOL to Isabelle/HOL [Obua 2006]
- Isabelle/HOL to HOL-Light [McLaughlin 2006]
- HOL-Light to Coq [Wiedijk 2007, no implementation]
- HOL-Light to Coq [Keller & Werner 2010]
- HOL-Light to HOL4 [Kumar 2013]
- HOL-Light to Metamath [Carneiro 2016]
- HOL4 to Isabelle/HOL [Immler et al 2019]
- Lean3 to Coq [Gilbert 2020]
- Lean3 to Lean4 [Lean community 2021]
- Maude to Lean [Rubio & Riesco 2022]
- ...

Interoperability between n systems ?



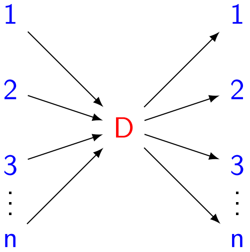
$n(n-1)$ translators

Interoperability between n systems ?



$n(n-1)$ translators

Can't we be more generic ?



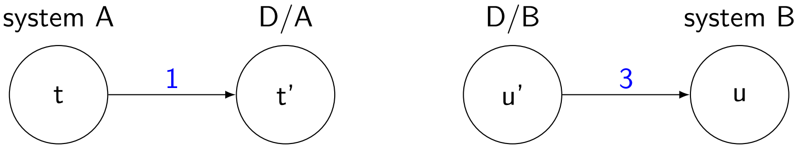
$2n$ translators

A common language for proofs?

A logical framework D

language for describing axioms, deduction rules and proofs of a system S as a theory D/S in D

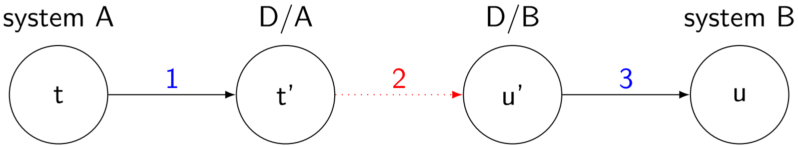
How to translate a proof $t \in A$ in a proof $u \in B$ in a logical framework D?



1. translate $t \in A$ in $t' \in D/A$

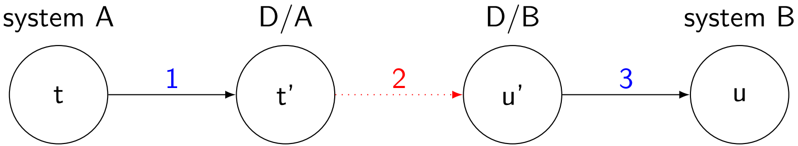
3. translate $u' \in D/B$ in $u \in B$

How to translate a proof $t \in A$ in a proof $u \in B$ in a logical framework D?



1. translate $t \in A$ in $t' \in D/A$
2. identify the axioms and deduction rules of A used in t'
   translate $t' \in D/A$ in $u' \in D/B$ if possible
3. translate $u' \in D/B$ in $u \in B$

How to translate a proof $t \in A$ in a proof $u \in B$ in a logical framework D?



1. translate $t \in A$ in $t' \in D/A$
2. identify the axioms and deduction rules of A used in t'
   translate $t' \in D/A$ in $u' \in D/B$ if possible
3. translate $u' \in D/B$ in $u \in B$

$\Rightarrow$ equally represent functionalities common to A and B

A common language for proofs?

A logical framework D

language for describing axioms, deduction rules and proofs of a system S as a theory D/S in D

Example: D = predicate calculus

allows one to represent S=geometry, S=arithmetic, S=set theory, ...
not well suited for computation and dependent types

A common language for proofs?

A logical framework D

language for describing axioms, deduction rules and proofs of a system S as a theory D/S in D

Example: $D =$ predicate calculus

allows one to represent S=geometry, S=arithmetic, S=set theory, ...
not well suited for computation and dependent types

Better: $D = \lambda\Pi$-calculus modulo rewriting/Dedukti

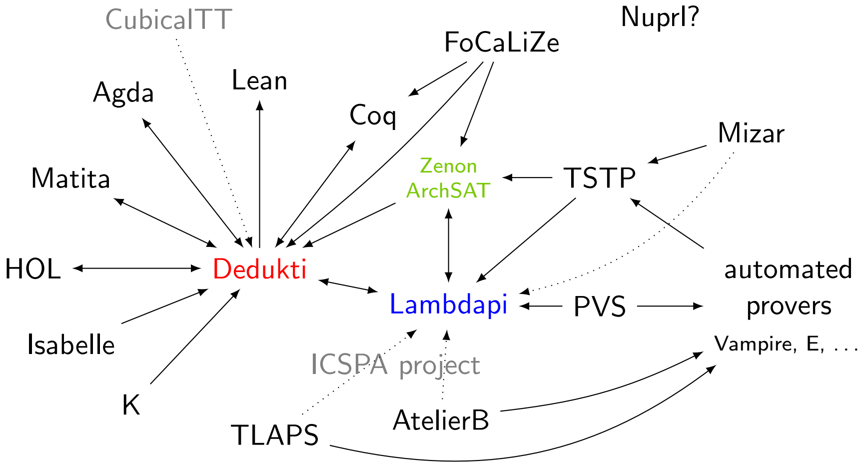
allows one to represent also:
S=HOL, S=Coq, S=Agda, S=PVS, ...

other options: λProlog, Twelf, Isabelle, Metamath, MMT...

The Dedukti world

- Zenon, ArchSAT, iProverModulo: ATPs generating Dedukti
- Holide: translator from OpenTheory to Dedukti
- Krajono: translator from Matita to Dedukti
- CoqInE: translator from Coq to Dedukti
- isabelle_dedukti: translator from Isabelle to Dedukti
- hol2dk: translator from HOL-Light to Dedukti and Lambdapi
- Agda2Dedukti: translator from Agda to Dedukti
- personoj: translator from PVS to Lambdapi
- ekstrakto: translator from TSTP to Lambdapi
- B-pog-translator: translator from Atelier B to Lambdapi
- sttfaxport: translator from Dedukti to OpenTheory, Matita, Coq, PVS and Lean3
- lambdapi: translator from Dedukti to Lambdapi, and from Lambdapi to Dedukti and Coq
- ...

Dedukti, an assembly language for proof systems



Lambdapi = Dedukti + implicit arguments/coercions, tactics, ...

https://github.com/Deducteam/Dedukti

https://github.com/Deducteam/lambdapi

Libraries translated to Dedukti

System	Libraries
OpenTheory	OpenTheory Library
HOL-Light	`hol.ml` NEW! (all ML files soon?)
Matita	Arithmetic Library
Coq	Stdlib parts, GeoCoq parts
Isabelle	HOL session, AFP parts NEW! (all AFP soon?)
Agda	Stdlib parts ($\pm$ 25%)
PVS	Stdlib parts (statements only)
TPTP	E 69%, Vampire 83% (for CNF only) integration in TPTP World via GDV NEW!

Libraries translated to Dedukti

System	Libraries
OpenTheory	OpenTheory Library
HOL-Light	`hol.ml` NEW! (all ML files soon?)
Matita	Arithmetic Library
Coq	Stdlib parts, GeoCoq parts
Isabelle	HOL session, AFP parts NEW! (all AFP soon?)
Agda	Stdlib parts ($\pm$ 25%)
PVS	Stdlib parts (statements only)
TPTP	E 69%, Vampire 83% (for CNF only) integration in TPTP World via GDV NEW!

Dedukti libraries can now be searched by using Lambdapi NEW!
See `https://lambdapi.readthedocs.io/` and

Claudio Sacerdoti Coen's **talk on Friday afternoon** at the EuroProofNet meeting at the Cambridge Computer Lab

Examples of translations via Dedukti

- Matita arith lib $\longrightarrow$ OpenTheory, Coq, PVS, Lean [Thiré 2018]
  http://logipedia.inria.fr
- Matita arith lib $\longrightarrow$ Agda [Felicissimo 2023] NEW!
  https://github.com/thiagofelicissimo/matita_lib_in_agda
- HOL-Light $\longrightarrow$ Coq NEW!
  https://github.com/Deducteam/hol2dk/
- Isabelle/HOL $\longrightarrow$ Coq NEW!
  https://github.com/Deducteam/isabelle_dedukti/
  [Dubut, Yamada, B., Leray, Färber, Wenzel]

Outline

Historical overview on proof system interoperability

How to encode logics in $\lambda\Pi/\mathcal{R}$?

Example: from HOL-Light to Coq via Lambdapi

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

$\lambda\Pi/\mathcal{R} =$	λ	simply-typed λ-calculus
	$+\ \Pi$	dependent types, e.g. Array n
	$+\ \mathcal{R}$	identification of types modulo rewrites rules $l \hookrightarrow r$

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

$\lambda\Pi/\mathcal{R} = \lambda$ simply-typed λ-calculus

$+ \Pi$ dependent types, e.g. Array n

$+ \mathcal{R}$ identification of types modulo rewrites rules $l \hookrightarrow r$

---

typing = typing of Edinburg's Logical Framework LF including:

$$\text{(abs)}\ \frac{\Gamma, x : A \vdash t : B \quad \Gamma \vdash \Pi x : A, B : \texttt{TYPE}}{\Gamma \vdash \lambda x : A, t : \Pi x : A, B}$$

$x \notin \Gamma$: types of local variables

$$\text{(app)}\ \frac{\Gamma \vdash t : \Pi x : A, B \quad \Gamma \vdash u : A}{\Gamma \vdash tu : B\{x \mapsto u\}}$$

+ the rule $$\text{(conv)}\ \frac{\Gamma \vdash t : A \quad A \equiv_{\beta\mathcal{R}} B}{\Gamma \vdash t : B}$$

$\equiv_{\beta\mathcal{R}}$: equational theory generated by β and $\mathcal{R}$

---

concat : $\Pi p : \mathbb{N}, \text{Array } p \rightarrow \Pi q : \mathbb{N}, \text{Array } q \rightarrow \text{Array}(p + q)$

concat 2 a 3 b : $\text{Array}(2 + 3) \equiv_{\beta\mathcal{R}} \text{Array}(5)$

First-order logic

- **the set of terms**
  built from a set of function symbols equipped with an arity
- **the set of propositions**
  built from a set of predicate symbols equipped with an arity
  and the logical connectives $\top$, $\bot$, $\neg$, $\Rightarrow$, $\wedge$, $\vee$, $\Leftrightarrow$, $\forall$, $\exists$
- **the set of axioms** (the actual theory)
- **the subset of provable propositions**
  using deduction rules, e.g. natural deduction:

$$(\Rightarrow\text{-intro})\ \frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \qquad (\Rightarrow\text{-elim})\ \frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}$$

$$(\forall\text{-intro})\ \frac{\Gamma \vdash A \quad x \notin \Gamma}{\Gamma \vdash \forall x, A} \qquad (\forall\text{-elim})\ \frac{\Gamma \vdash \forall x, A}{\Gamma \vdash A\{(x, u)\}}$$

...

Encoding of first-order logic

- **the set of terms** $\iota : \texttt{TYPE}$

  built from a set of function symbols equipped with an arity

  function symbol: $\iota \to \ldots \to \iota \to \iota$

Encoding of first-order logic

- **the set of terms** I : TYPE
  built from a set of function symbols equipped with an arity
  function symbol: $I \to \ldots \to I \to I$

- **the set of propositions** $Prop$: TYPE
  built from a set of predicate symbols equipped with an arity
  predicate symbol: $I \to \ldots \to I \to Prop$

Encoding of first-order logic

- **the set of terms** $I : \mathtt{TYPE}$
  built from a set of function symbols equipped with an arity
  function symbol: $I \to \ldots \to I \to I$

- **the set of propositions** $Prop : \mathtt{TYPE}$
  built from a set of predicate symbols equipped with an arity
  predicate symbol: $I \to \ldots \to I \to Prop$
  and the logical connectives $\top$, $\bot$, $\neg$, $\Rightarrow$, $\wedge$, $\vee$, $\Leftrightarrow$, $\forall$, $\exists$
  $\top : Prop$, $\neg : Prop \to Prop$, $\forall : (I \to Prop) \to Prop$, $\ldots$
  we use λ-calculus to encode quantifiers:
  we encode $\forall x, A$ as $\forall(\lambda x : I, A)$

Encoding of first-order logic

- **the set of terms** $I : \texttt{TYPE}$
  built from a set of function symbols equipped with an arity
  function symbol: $I \rightarrow \ldots \rightarrow I \rightarrow I$

- **the set of propositions** $Prop : \texttt{TYPE}$
  built from a set of predicate symbols equipped with an arity
  predicate symbol: $I \rightarrow \ldots \rightarrow I \rightarrow Prop$
  and the logical connectives $\top$, $\bot$, $\neg$, $\Rightarrow$, $\wedge$, $\vee$, $\Leftrightarrow$, $\forall$, $\exists$
  $\top : Prop$, $\neg : Prop \rightarrow Prop$, $\forall : (I \rightarrow Prop) \rightarrow Prop$, $\ldots$
  we use λ-calculus to encode quantifiers:
  we encode $\forall x, A$ as $\forall(\lambda x : I, A)$
  how to encode proofs?

- **the set of axioms** (the actual theory)

- **the subset of provable propositions**
  using deduction rules, e.g. natural deduction

Using λ-terms to represent proofs

(Curry-de Bruijn-Howard isomorphism)

logic	λ-calculus
proposition	type
proof	λ-term
proof checking	**type checking**
assumption	variable
$\Rightarrow$	$\rightarrow$
$\Rightarrow$-intro	abstraction
$\Rightarrow$-elim	application
$\forall$	Π
...	...

Using λ-terms to represent proofs

(Curry-de Bruijn-Howard isomorphism)

the natural deduction rules

$$(\Rightarrow\text{-intro})\ \frac{\Gamma, \quad A \vdash \quad B}{\Gamma \vdash \qquad A \Rightarrow B}$$

$$(\Rightarrow\text{-elim})\ \frac{\Gamma \vdash \quad A \Rightarrow B \quad \Gamma \vdash \quad A}{\Gamma \vdash \quad B}$$

$$(\forall\text{-intro})\ \frac{\Gamma \vdash \quad A \qquad x \notin \Gamma}{\Gamma \vdash \qquad \forall x, A}$$

$$(\forall\text{-elim})\ \frac{\Gamma \vdash \quad \forall x, A}{\Gamma \vdash \quad A\{(x, u)\}}$$

Using λ-terms to represent proofs

(Curry-de Bruijn-Howard isomorphism)

by giving a name to every assumption, we get a typing environment

$$A_1, \ldots, A_n \quad \rightsquigarrow \quad x_1 : A_1, \ldots, x_n : A_n$$

by mapping every deduction rule to a λ-term construction
the typing rules of $\lambda\Pi$ correspond to the natural deduction rules

$$(\Rightarrow\text{-intro})\ \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A, t : A \Rightarrow B}$$

$$(\Rightarrow\text{-elim})\ \frac{\Gamma \vdash t : A \Rightarrow B \quad \Gamma \vdash u : A}{\Gamma \vdash tu : B}$$

$$(\forall\text{-intro})\ \frac{\Gamma \vdash t : A \qquad x \notin \Gamma}{\Gamma \vdash \lambda x, t : \forall x, A}$$

$$(\forall\text{-elim})\ \frac{\Gamma \vdash t : \forall x, A}{\Gamma \vdash tu\ : A\{(x, u)\}}$$

Encoding the Curry-de Bruijn-Howard isomorphism

terms of type *Prop* are not types...

but we can interpret a proposition as a type by taking:

$$\mathit{Prf} : \mathit{Prop} \to \texttt{TYPE}$$

Prf A is the type of proofs of proposition A

Encoding the Curry-de Bruijn-Howard isomorphism

terms of type *Prop* are not types...

but we can interpret a proposition as a type by taking:

$$\mathit{Prf} : \mathit{Prop} \rightarrow \mathtt{TYPE}$$

Prf A is the type of proofs of proposition A

but

$$\lambda x : \mathit{Prf}\ A, x \quad : \quad \mathit{Prf}\ A \rightarrow \mathit{Prf}\ A$$

and

$$\lambda x : \mathit{Prf}\ A, x \quad \not: \quad \mathit{Prf}\,(A \Rightarrow A)$$

Encoding the Curry-de Bruijn-Howard isomorphism

terms of type *Prop* are not types...

but we can interpret a proposition as a type by taking:

$$Prf : Prop \to \mathtt{TYPE}$$

Prf A is the type of proofs of proposition A

but

$$\lambda x : Prf\ A, x \quad : \quad Prf\ A \to Prf\ A$$

and

$$\lambda x : Prf\ A, x \quad \not: \quad Prf\,(A \Rightarrow A)$$

unless we add the <u>rewrite rule</u>

$$Prf\,(A \Rightarrow B) \quad \hookrightarrow \quad Prf\ A \to Prf\ B$$

Encoding $\Rightarrow$

because $\mathit{Prf}\,(A \Rightarrow B) \hookrightarrow \mathit{Prf}\ A \to \mathit{Prf}\ B$

the introduction rule for $\Rightarrow$ is the abstraction:

$$(\Rightarrow\text{-intro})\ \frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \qquad \begin{array}{l} (\text{abs})\ \dfrac{\Gamma, x : \mathit{Prf}\ A \vdash t : \mathit{Prf}\ B}{\Gamma \vdash \lambda x : A, t : \mathit{Prf}\ A \to \mathit{Prf}\ B} \\ (\text{conv})\ \dfrac{}{\Gamma \vdash \lambda x : A, t : \mathit{Prf}\,(A \Rightarrow B)} \end{array}$$

Encoding $\Rightarrow$

because $\mathit{Prf}(A \Rightarrow B) \hookrightarrow \mathit{Prf}\ A \rightarrow \mathit{Prf}\ B$

the introduction rule for $\Rightarrow$ is the abstraction:

$$(\Rightarrow\text{-intro})\ \frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \qquad \text{(conv)}\ \frac{\text{(abs)}\ \dfrac{\Gamma, x : \mathit{Prf}\ A \vdash t : \mathit{Prf}\ B}{\Gamma \vdash \lambda x : A, t : \mathit{Prf}\ A \rightarrow \mathit{Prf}\ B}}{\Gamma \vdash \lambda x : A, t : \mathit{Prf}(A \Rightarrow B)}$$

the elimination rule for $\Rightarrow$ is the application:

$$(\Rightarrow\text{-elim})\ \frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}$$

$$\text{(app)}\ \frac{\text{(conv)}\ \dfrac{\Gamma \vdash t : \mathit{Prf}(A \Rightarrow B)}{\Gamma \vdash t : \mathit{Prf}\ A \rightarrow \mathit{Prf}\ B} \quad \Gamma \vdash u : \mathit{Prf}\ A}{\Gamma \vdash tu : \mathit{Prf}\ B}$$

Encoding $\forall$

we can do something similar for $\forall : (I \to Prop) \to Prop$ by taking:

$$Prf\,(\forall A) \quad \hookrightarrow \quad \Pi x : I, Prf\,(A\,x)$$

then the introduction rule for $\forall$ is the abstraction
and the elimination rule for $\forall$ is the application

Encoding the other connectives

the other connectives can be defined
by using a meta-level quantification on propositions:

$$Prf\,(A \wedge B) \quad \hookrightarrow \quad \Pi C : Prop, (Prf\ A \to Prf\ B \to Prf\ C) \to Prf\ C$$

Encoding the other connectives

the other connectives can be defined
by using a meta-level quantification on propositions:

$$\mathit{Prf}(A \wedge B) \quad \hookrightarrow \quad \Pi C : \mathit{Prop}, (\mathit{Prf}\ A \to \mathit{Prf}\ B \to \mathit{Prf}\ C) \to \mathit{Prf}\ C$$

introduction and elimination rules can be derived:

($\wedge$-intro):

$$\lambda a : \mathit{Prf}\ A, \lambda b : \mathit{Prf}\ B, \lambda C : \mathit{Prop}, \lambda h : \mathit{Prf}\ A \to \mathit{Prf}\ B \to \mathit{Prf}\ C, hab$$

is of type

$$\mathit{Prf}\ A \to \mathit{Prf}\ B \to \mathit{Prf}(A \wedge B)$$

($\wedge$-elim1):

$$\lambda c : \mathit{Prf}(A \wedge B), c\, A\, (\lambda a : \mathit{Prf}\ A, \lambda b : \mathit{Prf}\ B, a)$$

is of type

$$\mathit{Prf}(A \wedge B) \to \mathit{Prf}\ A$$

To summarize: $\lambda\Pi/\mathcal{R}$-theory *FOL* for first-order logic

signature Σ_{FOL}:

$I : \texttt{TYPE}$

$f : I \to \ldots \to I \to I$ for each function symbol f of arity n

$Prop : \texttt{TYPE}$

$P : I \to \ldots \to I \to Prop$ for each predicate symbol P of arity n

$\top : Prop$, $\neg : Prop \to Prop$, $\forall : (I \to Prop) \to Prop$, ...

$Prf : Prop \to \texttt{TYPE}$

$a : Prf\ A$ for each axiom A

rules $\mathcal{R}_{FOL}$:

$$
\begin{aligned}
Prf(A \Rightarrow B) &\hookrightarrow Prf\ A \to Prf\ B \\
Prf(\forall A) &\hookrightarrow \Pi x : I, Prf(A\,x) \\
Prf(A \wedge B) &\hookrightarrow \Pi C : Prop, (Prf\ A \to Prf\ B \to Prf\ C) \to Prf\ C \\
Prf\,\bot &\hookrightarrow \Pi C : Prop, Prf\ C \\
Prf(\neg A) &\hookrightarrow Prf\,A \to Prf\,\bot
\end{aligned}
$$

...

Encoding of first-order logic in $\lambda\Pi/FOL$

encoding of terms:

$$|x| = x$$
$$|ft_1 \dots t_n| = f|t_1| \dots |t_n|$$

encoding of propositions:

$$|Pt_1 \dots t_n| = P|t_1| \dots |t_n|$$
$$|\top| = \top$$
$$|A \wedge B| = |A| \wedge |B|$$
$$|\forall x, A| = \forall(\lambda x : I, |A|)$$

...

$$|\Gamma, A| = |\Gamma|, x_{\|\Gamma\|+1} : A$$

encoding of proofs:

$$\left| \frac{\pi_{\Gamma, A \vdash B}}{\Gamma \vdash A \Rightarrow B} (\Rightarrow_i) \right| = \lambda x_{\|\Gamma\|+1} : Prf\ |A|, |\pi_{\Gamma, A \vdash B}|$$

$$\left| \frac{\pi_{\Gamma \vdash A \Rightarrow B} \quad \pi_{\Gamma \vdash A}}{\Gamma \vdash B} (\Rightarrow_e) \right| = |\pi_{\Gamma \vdash A \Rightarrow B}|\,|\pi_{\Gamma \vdash A}|$$

...

Properties of the encoding in $\lambda\Pi/FOL$

- a term is mapped to a term of type I
- a proposition is mapped to a term of type $Prop$
- a proof of A is mapped to a term of type $Prf\ |A|$

Properties of the encoding in $\lambda\Pi/FOL$

- a term is mapped to a term of type I
- a proposition is mapped to a term of type $Prop$
- a proof of A is mapped to a term of type $Prf\ |A|$

if we find t of type $Prf\ |A|$, can we deduce that A is provable ?

Properties of the encoding in $\lambda\Pi/FOL$

- a term is mapped to a term of type I
- a proposition is mapped to a term of type $Prop$
- a proof of A is mapped to a term of type $Prf\ |A|$

if we find t of type $Prf\ |A|$, can we deduce that A is provable ?

- yes, the encoding is **conservative:**
  if $Prf\ |A|$ is inhabited then A is provable

proof sketch: because $\hookrightarrow_{\beta\mathcal{R}}$ terminates and is confluent, t has a normal form, and terms in normal form can be easily translated back in first-order logic and natural deduction

Multi-sorted first-order logic

for each sort I_k (e.g. point, line, circle), add:

$I_k : \mathtt{TYPE}$
$\forall_k : (I_k \to Prop) \to Prop$

$Prf(\forall_k A) \hookrightarrow \Pi x : I_k, Prf(Ax)$

Polymorphic first-order logic

same trick as Curry-de Bruijn-Howard

$Set : \texttt{TYPE}$
$El : Set \to \texttt{TYPE}$
$\iota : Set$ for each sort ι
$\forall : \Pi a : Set, (El\, a \to Prop) \to Prop$

$Prf\,(\forall a p) \hookrightarrow \Pi x : El\, a, Prf\,(p\, x)$

Higher-order logic

order	quantification on
1	elements
2	sets of elements
3	sets of sets of elements
...	...
ω	any set

Higher-order logic

order	quantification on
1	elements
2	sets of elements
3	sets of sets of elements
...	...
ω	any set

quantification on functions:

$\leadsto : Set \to Set \to Set$

$El(a \leadsto b) \hookrightarrow El\ a \to El\ b$

Higher-order logic

order	quantification on
1	elements
2	sets of elements
3	sets of sets of elements
...	...
ω	any set

quantification on functions:

$\leadsto\ : \mathit{Set} \to \mathit{Set} \to \mathit{Set}$

$\mathit{El}(a \leadsto b) \hookrightarrow \mathit{El}\ a \to \mathit{El}\ b$

quantification on propositions/impredicativity (e.g. $\forall p, p \Rightarrow p$):

$o : \mathit{Set}$

$\mathit{El}\ o \hookrightarrow \mathit{Prop}$

Encoding dependent constructions

dependent implication:

$\Rightarrow_d : \Pi a : Prop, (Prf\ a \to Prop) \to Prop$

$Prf(a \Rightarrow_d b) \hookrightarrow \Pi x : Prf\ a, Prf(b\,x)$

Encoding dependent constructions

dependent implication:

$\Rightarrow_d : \Pi a : Prop, (Prf\ a \to Prop) \to Prop$

$Prf\,(a \Rightarrow_d b) \hookrightarrow \Pi x : Prf\ a, Prf\,(b\,x)$

dependent types:

$\leadsto_d : \Pi a : Set, (El\ a \to Set) \to Set$

$El(a \leadsto_d b) \hookrightarrow \Pi x : El\ a, El(b\,x)$

Encoding dependent constructions

dependent implication:

$\Rightarrow_d : \Pi a : Prop, (Prf\ a \to Prop) \to Prop$

$Prf(a \Rightarrow_d b) \hookrightarrow \Pi x : Prf\ a, Prf(b\, x)$

dependent types:

$\leadsto_d : \Pi a : Set, (El\ a \to Set) \to Set$

$El(a \leadsto_d b) \hookrightarrow \Pi x : El\ a, El(b\, x)$

proofs in object-terms:

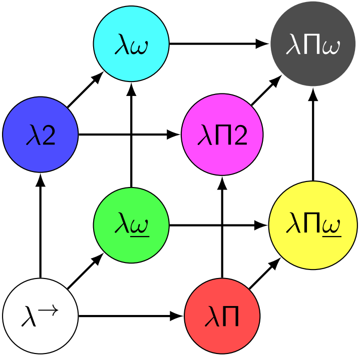
$\pi : \Pi p : Prop, (Prf\ p \to Set) \to Set$

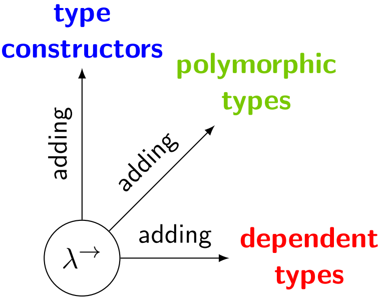
$El(\pi\ p\ a) \hookrightarrow \Pi x : Prf\ p, El(a\, x)$

example: $div : El(\iota \leadsto \iota \leadsto_d \lambda y : El\ \iota, \pi(y > 0)(\lambda\_, \iota))$
takes 3 arguments: $x : El\ \iota$, $y : El\ \iota$, $p : Prf(y > 0)$
and returns a term of type $El\ \iota$

Encoding the systems of Barendregt's λ-cube

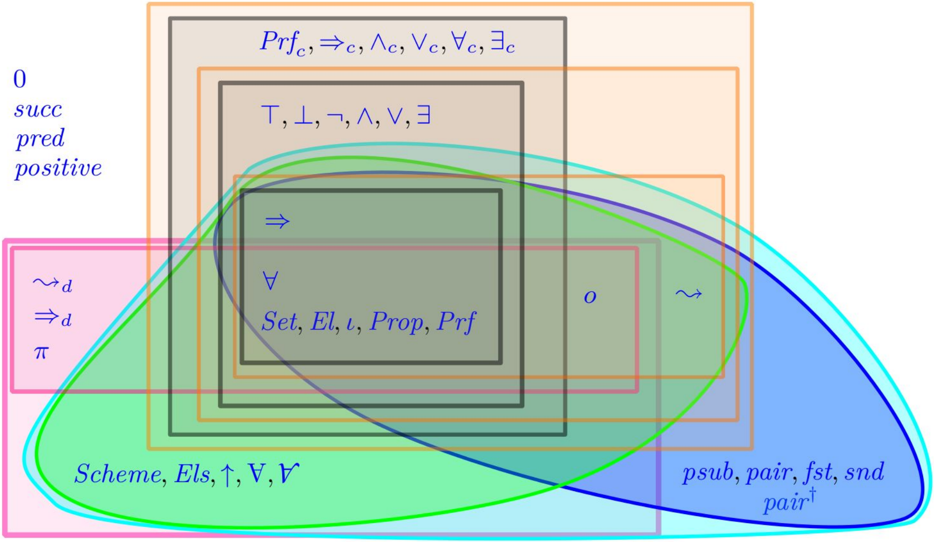
system	**PTS rule**	$\lambda\Pi/\mathcal{R}$ **rule**
simple types	$\mathtt{TYPE}, \mathtt{TYPE}$	$Prf\,(a \Rightarrow_d b) \hookrightarrow \Pi x : Prf\ a, Prf\,(b\,x)$
polymorphic types	$\mathtt{KIND}, \mathtt{TYPE}$	$Prf\,(\forall a b) \hookrightarrow \Pi x : El\ a, Prf\,(b\,x)$
dependent types	$\mathtt{TYPE}, \mathtt{KIND}$	$El(\pi\ a\ b) \hookrightarrow \Pi x : Prf\ a, El(b\,x)$
type constructors	$\mathtt{KIND}, \mathtt{KIND}$	$El(a \rightsquigarrow_d b) \hookrightarrow \Pi x : El\ a, El(b\,x)$





The modular $\lambda\Pi/\mathcal{R}$ theory U and its sub-theories

[B., Dowek, Grienenberger, Hondet, Thiré 2021]



Lambdapi files

Functional Pure Type Systems $(\mathcal{S}, \mathcal{A}, \mathcal{P})$ $\mathcal{A} \subseteq \mathcal{S}^2, \mathcal{P} \subseteq \mathcal{S}^2 \times \mathcal{S}$

terms and types:

$$t := x \mid tt \mid \lambda x : t, t \mid \Pi x : t, t \mid s \in \mathcal{S}$$

typing rules:

$$\frac{}{\emptyset \vdash} \qquad \frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash} \qquad \frac{\Gamma \vdash \quad (x, A) \in \Gamma}{\Gamma \vdash x : A}$$

$$(sort)\ \frac{\Gamma \vdash \quad (s_1, s_2) \in \mathcal{A}}{\Gamma \vdash s_1 : s_2}$$

$$(prod)\ \frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2 \quad ((s_1, s_2), s_3) \in \mathcal{P}}{\Gamma \vdash \Pi x : A, B : s_3}$$

$$\frac{\Gamma, x : A \vdash t : B \quad \Gamma \vdash \Pi x : A, B : s}{\Gamma \vdash \lambda x : A, t : \Pi x : A, B} \qquad \frac{\Gamma \vdash t : \Pi x : A, B \quad \Gamma \vdash u : A}{\Gamma \vdash tu : B\{(x, u)\}}$$

$$\frac{\Gamma \vdash t : A \quad A \simeq_\beta A' \quad \Gamma \vdash A' : s}{\Gamma \vdash t : A'}$$

Encoding Functional Pure Type Systems

[Cousineau & Dowek 2007]

signature:

$U_s : \mathtt{TYPE}$ for each sort $s \in \mathcal{S}$

$El_s : U_s \to \mathtt{TYPE}$

$s_1 : U_{s_2}$ for every $(s_1, s_2) \in \mathcal{A}$

$\pi_{s_1,s_2} : \Pi a : U_{s_1}, (El_{s_1}\, a \to U_{s_2}) \to U_{s_3}$ for every $((s_1, s_2), s_3) \in \mathcal{P}$

rules:

$El_{s_2}\, s_1 \hookrightarrow U_{s_1}$ for every $(s_1, s_2) \in \mathcal{A}$

$El_{s_3}(\pi_{s_1,s_2}\, a\, b) \hookrightarrow \Pi x : El_{s_1}\, a, El_{s_2}(b\, x)$ for every $((s_1, s_2), s_3) \in \mathcal{P}$

encoding:

$|x|_\Gamma = x$

$|s|_\Gamma = s$

$|\lambda x : A, t|_\Gamma = \lambda x : El_s |A|_\Gamma, |t|_{\Gamma, x:A}$ if $\Gamma \vdash A : s$

$|tu|_\Gamma = |t|_\Gamma |u|_\Gamma$

$|\Pi x : A, B|_\Gamma = \pi_{s_1,s_2} |A|_\Gamma (\lambda x : El_{s_1} |A|_\Gamma, |B|_{\Gamma, x:A})$

if $\Gamma \vdash A : s_1$ and $\Gamma, x : A \vdash B : s_2$

Encoding other features

- recursive functions [Assaf 2015, Cauderlier 2016, Férey 2021]
  - different approaches, no general theory
  - encoding in recursors [ongoing work by Felicissimo & Cockx]
- universe polymorphism [Genestier 2020]
  - requires rewriting with matching modulo AC
    or rewriting on AC canonical forms [B. 2022]
- η-conversion on function types [Genestier 2020]
- predicate subtyping with proof irrelevance [Hondet 2020]
- co-inductive objects and co-recursion [Felicissimo 2021]

Outline

Historical overview on proof system interoperability

How to encode logics in $\lambda\Pi/\mathcal{R}$?

Example: from HOL-Light to Coq via Lambdapi

Previous works & tools on HOL to Coq

- **Denney 2000:** translates HOL98 proofs [Wong 1999] to Coq scripts using some intermediate stack-based machine language
- **Wiedijk 2007:** describes a translation of HOL-Light logic and proofs in Coq terms via shallow embedding (no implementation)
- **Keller & Werner 2010:** translates HOL-Light proofs [Obua & Skalberg 2006] to Coq terms via deep embedding & computational reflection (but no automatic shallow embedding)

Previous works & tools on HOL to Coq

- **Denney 2000:** translates HOL98 proofs [Wong 1999] to Coq scripts using some intermediate stack-based machine language
- **Wiedijk 2007:** describes a translation of HOL-Light logic and proofs in Coq terms via shallow embedding (no implementation)
- **Keller & Werner 2010:** translates HOL-Light proofs [Obua & Skalberg 2006] to Coq terms via deep embedding & computational reflection (but no automatic shallow embedding)
- **B. 2023:** implements Wiedijk approach to translate HOL-Light proofs [Polu 2019] to Coq via a shallow embedding in Lambdapi

Converting HOL-Light proofs to Coq via Lambdapi

- https://github.com/Deducteam/hol2dk
  - provides a small patch for HOL-Light to export proofs

    improves ProofTrace [Polu 2019] by reducing memory consumption and adding on-the-fly writing on disk
  - translates HOL-Light proofs to Dedukti and Lambdapi
- https://github.com/Deducteam/lambdapi
  - allows to converts dk/lp files using some encodings of HOL into Coq files

HOL-Light logic

$$\frac{}{\vdash t = t}\ \text{REFL} \qquad \frac{\Gamma \vdash s = t \quad \Delta \vdash t = u}{\Gamma \cup \Delta \vdash s = u}\ \text{TRANS}$$

$$\frac{\Gamma \vdash s = t \quad \Delta \vdash u = v}{\Gamma \cup \Delta \vdash su = tv}\ \text{MK\_COMB} \qquad \frac{\Gamma \vdash s = t}{\lambda x, s = \lambda x, t}\ \text{ABS}$$

$$\frac{}{\vdash (\lambda x, t)x = t}\ \text{BETA} \qquad \frac{}{\{p\} \vdash p}\ \text{ASSUME}$$

$$\frac{\Gamma \vdash p = q \quad \Delta \vdash p}{\Gamma \cup \Delta \vdash q}\ \text{EQ\_MP}$$

$$\frac{\Gamma \vdash p \quad \Delta \vdash q}{(\Gamma - \{q\}) \cup (\Delta - \{p\}) \vdash p = q}\ \text{DEDUCT\_ANTISYM\_RULE}$$

$$\frac{\Gamma \vdash p}{\Gamma\theta \vdash p\theta}\ \text{INST} \qquad \frac{\Gamma \vdash p}{\Gamma\Theta \vdash p\Theta}\ \text{INST\_TYPE}$$

HOL-Light logic: connectives are defined from equality!

$$\top =_{def} (\lambda p.p) = (\lambda p.p)$$
$$\wedge =_{def} \lambda p.\lambda q.(\lambda f.fpq) = (\lambda f.f\top\top)$$
$$\Rightarrow =_{def} \lambda p.\lambda q.(p \wedge q) = p$$
$$\forall =_{def} \lambda p.p = (\lambda x.\top)$$
$$\exists =_{def} \lambda p.\forall q.(\forall x.px \Rightarrow q) \Rightarrow q$$
$$\vee =_{def} \lambda p.\lambda q.\forall r.(p \Rightarrow r) \Rightarrow (q \Rightarrow r) \Rightarrow r$$
$$\bot =_{def} \forall p.p$$
$$\neg =_{def} \lambda p.p \Rightarrow \bot$$

Example: `hol.ml` (HOL-Light standard library)

```
loads "pair.ml";;         (* Theory of pairs
loads "compute.ml";;      (* General call-by-value reduction tool for ter
loads "nums.ml";;         (* Axiom of Infinity, definition of natural num
loads "recursion.ml";;    (* Tools for primitive recursion on inductive t
loads "arith.ml";;        (* Natural number arithmetic
loads "wf.ml";;           (* Theory of wellfounded relations
loads "calc_num.ml";;     (* Calculation with natural numbers
loads "normalizer.ml";;   (* Polynomial normalizer for rings and semiring
loads "grobner.ml";;      (* Groebner basis procedure for most semirings
loads "ind_types.ml";;    (* Tools for defining inductive types
loads "lists.ml";;        (* Theory of lists
loads "realax.ml";;       (* Definition of real numbers
loads "calc_int.ml";;     (* Calculation with integer-valued reals
loads "realarith.ml";;    (* Universal linear real decision procedure
loads "real.ml";;         (* Derived properties of reals
loads "calc_rat.ml";;     (* Calculation with rational-valued reals
loads "int.ml";;          (* Definition of integers
loads "sets.ml";;         (* Basic set theory
loads "iterate.ml";;      (* Iterated operations
loads "cart.ml";;         (* Finite Cartesian products
loads "define.ml";;       (* Support for general recursive definitions
```

Results for `hol.ml` by instrumenting rules only

- number of theorems: 2834
- number of proof steps: 14.3 M
- proof file size: 5.5 Go
- checking time by OCaml without proof generation: 1m14s
- checking time by OCaml with proof generation: 2m9s (+74%)

rule	% steps
refl	26
eqmp	21
term-subst	15
trans	11
mk-comb	10
deduct	7
type-subst	4
abs	2
beta	2
assume	2

Reducing proof size by instrumenting basic tactics

- introduction/elimination rules of connectives
- alpha conversion (20% of proof steps!)

instrumenting

	rules only	connectives,alpha	variation
steps	14.3 M	8.9 M	-38%
size	5.5 Go	3.1 Go	-44%

Reducing proof size by instrumenting basic tactics

- introduction/elimination rules of connectives
- alpha conversion (20% of proof steps!)

instrumenting

	rules only	connectives,alpha	variation
steps	14.3 M	8.9 M	-38%
size	5.5 Go	3.1 Go	-44%

% steps

rule	rules only	connectives,alpha	variation
refl	26	29	+3
eqmp	21	19	-2
term-subst	15	12	-3
trans	11	6	-5
mk-comb	10	17	+7
deduct	7	1	-6
type-subst	4	3	-1
abs	2	2	0
beta	2	3	+1
assume	2	1	-1

Translation of `hol.ml` to Dedukti and Lambdapi

HOL-Light proof file: 3.1 Go (8.9 M proof steps)

the translation can be done in parallel:

	dk	lp
size	3.3 Go	2.2 Go
time 1 thread	22m37s	12m8s
time 7 threads	9m2s	4m23s

Checking generated Dedukti files

the obtained Dedukti files are big (3.3 Go)

but can be checked in **12m52s** by kocheck:

Safe, fast, concurrent proof checking for the lambda-pi calculus modulo rewriting, M. Färber, CPP'22

lambdapi is too slow and requires too much memory

Translation of HOL to Coq

HOL proofs can be translated to Coq using the following axioms:

- Indefinite description/Hilbert ε:
  `forall A (P:A->Prop), (exists x, P x) -> {x : A | P x}`
- Functional extensionnality:
  `forall A B (f g:A -> B), (forall x, f x = g x) -> f = g`
- Propositional extensionnality:
  `forall (P Q:Prop), (P -> Q) -> (Q -> P) -> P = Q`

and by mapping:

- HOL-Light types to Coq non-empty types (canonical structure)
- HOL-Light bool type to Coq type of propositions
- HOL-Light natural numbers to Coq natural numbers
- HOL-Light connectives to Coq connectives
- HOL-Light equality to Coq equality
- ...

Translation of Lambdapi/HOL to Coq

Lambdapi can translate dk/lp files using HOL encodings to Coq

Example: lp files obtained from `hol.ml`

- lp files size: 2.2 Go
- translation to Coq: 2m22s
- coq files size: 2.1 Go

but Coq requires several hours to check those files on a powerful machine (RAM $>$ 32 Go required)

A smaller example: HOL-Light basic arithmetic library

proof dumping	11.7s, 82 Mo, 324 K proof steps
dk file generation	6.6s, 82 Mo
checking time with dk check	13.6s
lp file generation	3.7s, 56 Mo
checking time with lambdapi	1m22s
translation to Coq	2.8s, 52 Mo
checking time with Coq 8.17.1	4m

example output:

```
Lemma thm_DIV_DIV : forall m : nat, forall n : nat,
  forall p : nat, (DIV (DIV m n) p) = (DIV m (mul n p)).

Lemma thm_DIV_MOD : forall m : nat, forall n : nat,
  forall p : nat, (MOD (DIV m n) p) = (DIV (MOD m (mul n p)
```

TODO on `hol2dk`

- comparison with previous work difficult since their code is lost or not (easily) working anymore (they are not maintained)
- instrument symmetry, definition unfoldings and rewrite tactics to reduce the size of proofs further
- map each ML file to a dk/lp file
- make dk/lp translation incremental

Conclusion

- interoperability theory/tools developed for 30 years now but few tools are really usable for lack of maintenance
- significant progresses have been done on genericity by using the $\lambda\Pi$-calculus modulo rewriting/Dedukti
- works well for medium-size developments with simple structures (integers, lists, ...) and automated theorem provers, e.g. integration of Lambdapi in TPTP World/GDV [Sutcliffe] NEW!
- some people are skeptikal on the usability of translations on complex structures but some progress is ongoing, e.g. translation of type classes between Isabelle & Coq [Sacerdoti & Tassi] NEW!
- improving scalability, modularity, usability and reproducibility are exciting research problems!