Proof Systems Interoperability

Frédéric Blanqui

EuroProofNet

Outline

Historical overview on proof systems interoperability

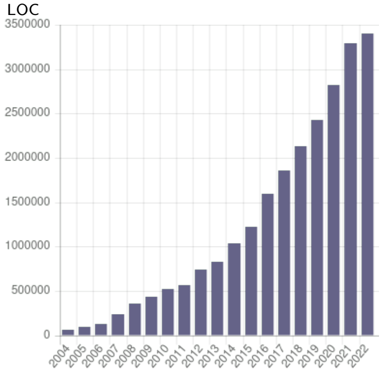
How to encode logics in $\lambda\Pi/\mathcal{R}$?

Example: from HOL-Light to Coq via Lambdapi

Libraries of formal proofs today

Library	Nb files	Nb objects*
Coq Opam	37,700	1,285,000
Isabelle AFP	8,700	272,000
Lean Mathlib	4,600	238,000
Mizar Mathlib	1,400	77,000
HOL-Light Lib	635	36,500
...	...	...

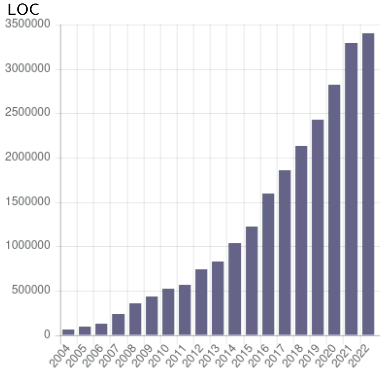
* type, definition, theorem, ... (July 2024)



Libraries of formal proofs today

Library	Nb files	Nb objects*
Coq Opam	37,700	1,285,000
Isabelle AFP	8,700	272,000
Lean Mathlib	4,600	238,000
Mizar Mathlib	1,400	77,000
HOL-Light Lib	635	36,500
...	...	...

* type, definition, theorem, ... (July 2024)



- 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 systems 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 systems 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 proof using impredicativity or proof irrelevance in a system not allowing these features)

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]
  OpenTheory proof format for HOL-based proof assistants
- 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]

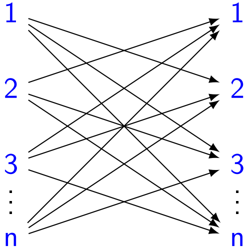
One-to-one translation tools

- 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]

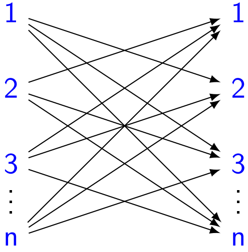
- HOL to Isabelle/HOL [Obua 2006]
- Isabelle/HOL to HOL-Light [McLaughlin 2006]
- HOL-Light to Coq [Wiedijk 2007, no implementation]
- HOL-Light to Coq [Keller & Werner 2010]
- HOL-Light to HOL4 [Kumar 2013]
- HOL-Light to Metamath [Carneiro 2016]
- HOL4 to Isabelle/HOL [Immler et al 2019]
- Lean3 to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

Interoperability between n systems ?

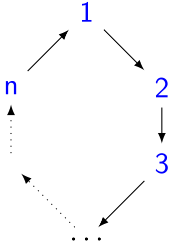


$n(n-1)$ translators

Interoperability between n systems ?

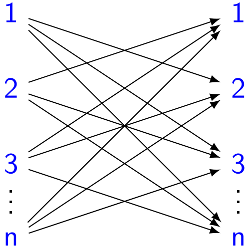


$n(n-1)$ translators

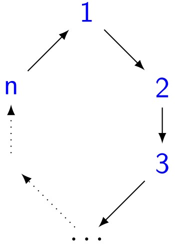


n translators

Interoperability between n systems ?

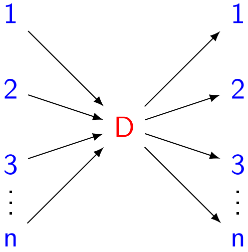


$n(n-1)$ translators



n translators

Can't we be more generic ?



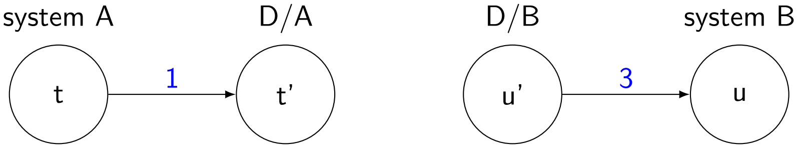
$2n$ translators

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

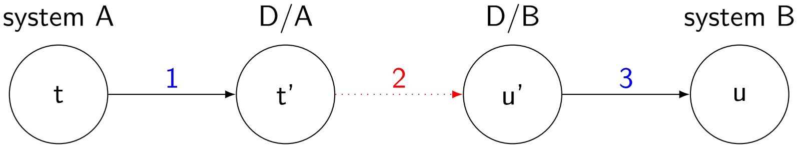
How to translate a proof $t \in A$ in a proof $u \in B$ via a logical framework D?



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

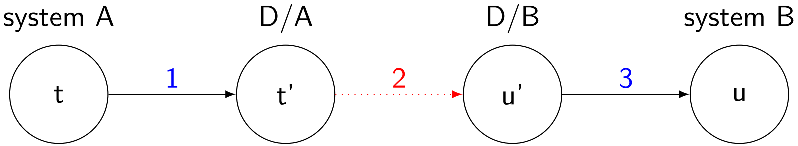
3. translate $u' \in D/B$ in $u \in B$

How to translate a proof $t \in A$ in a proof $u \in B$ via 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 $u \in B$ via 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 $u \in B$

$\Rightarrow$ represent features common to A & B identically in D/A & D/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

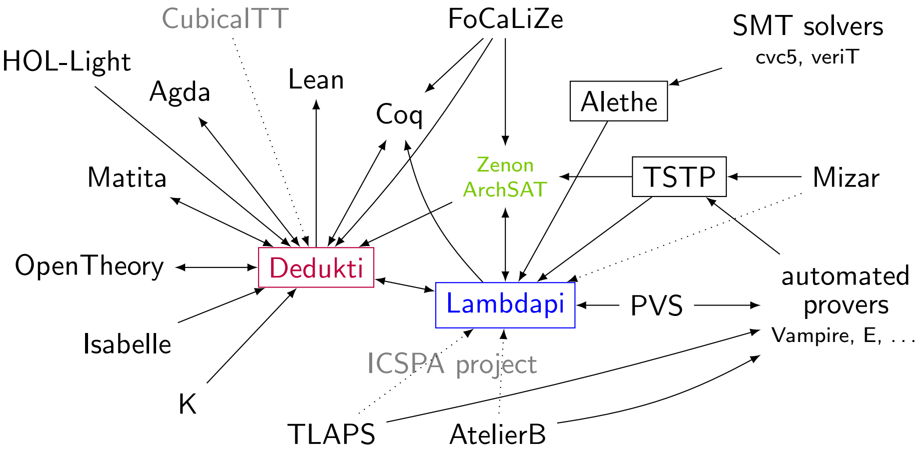
allows one to represent S=geometry, S=arithmetic, S=set theory, ...
not well suited for computation and dependent types

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, ...

Dedukti, an assembly language for proof systems



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

All translation tools are available on `https://github.com/Deducteam/`

Libraries translated to Dedukti

System	Libraries
OpenTheory	OpenTheory Library
HOL-Light	`Multivariate` NEW! (all ML files soon?)
Matita	Arithmetic Library
Coq	Stdlib parts, GeoCoq parts
Isabelle	HOL session, AFP parts NEW! (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	`Multivariate` NEW! (all ML files soon?)
Matita	Arithmetic Library
Coq	Stdlib parts, GeoCoq parts
Isabelle	HOL session, AFP parts NEW! (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!

Remark: Dedukti libraries can be searched by using Lambdapi index and search commands (Claudio Sacerdoti Coen)

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]
  https://github.com/thiagofelicissimo/matita_lib_in_agda
- HOL-Light $\longrightarrow$ Coq [B. 2024]
  https://github.com/Deducteam/hol2dk/
- Isabelle/HOL $\longrightarrow$ Coq (work in progress)
  [B., Dubut, Yamada, Leray, Färber, Wenzel]
  https://github.com/Deducteam/isabelle_dedukti/

Outline

Historical overview on proof systems 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:

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

$x \notin \Gamma$: 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 : $\Pi p : \mathbb{N}, \text{Array } p \to \Pi q : \mathbb{N}, \text{Array } q \to \text{Array}(p+q)$

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

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

- **the set of propositions** $\quad Prop : \texttt{TYPE}$
  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** $I : \mathtt{TYPE}$
  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 $\top$, $\bot$, $\neg$, $\Rightarrow$, $\wedge$, $\vee$, $\Leftrightarrow$, $\forall$, $\exists$
  $\top : Prop$, $\neg : Prop \to Prop$, $\forall : (I \to Prop) \to Prop$, ...
  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)

by interpreting propositions as types ($\Rightarrow/\rightarrow$, $\forall/\Pi$)

the typing rules of $\lambda\Pi$ correspond to the rules of natural deduction:

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

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

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

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

and proof checking is reduced to type checking

Expliciting the Brouwer-Heyting-Kolmogorov interpretation

terms of type Prop are not types...

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

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

$\mathit{Prf}\ A$ is the type of proofs of proposition A

Expliciting the Brouwer-Heyting-Kolmogorov interpretation

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

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

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

$Prf\ A$ is the type of proofs of proposition A

but

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

and

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

Expliciting the Brouwer-Heyting-Kolmogorov interpretation

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

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

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

$Prf\ A$ 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>

$$\boxed{Prf(A \Rightarrow B) \quad \hookrightarrow \quad Prf\ A \to Prf\ B}$$

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:

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

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:

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

Encoding the other connectives

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

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

introduction and elimination rules can be derived:

($\wedge$-intro):

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

is of type

$$Prf\ A \to Prf\ B \to Prf\,(A \wedge B)$$

($\wedge$-elim1):

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

is of type

$$Prf\,(A \wedge B) \to Prf\ A$$

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:

$$|x| = x$$
$$|ft_1 \ldots t_n| = f\,|t_1| \ldots |t_n|$$

encoding of propositions:

$$|Pt_1 \ldots t_n| = P\,|t_1| \ldots |t_n|$$
$$|\top| = \top$$
$$|A \wedge B| = |A| \wedge |B|$$
$$|\forall x, A| = \forall(\lambda x : I, |A|)$$
$$\ldots$$
$$|\Gamma, A| = |\Gamma|, x_{\|\Gamma\|+1} : A$$

encoding of proofs:

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

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

$\ldots$

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 for the BHK interpretation of propositions:

$Set : \mathtt{TYPE}$	type of sorts
$El : Set \to \mathtt{TYPE}$	interpretation of sorts as types
$\iota : Set$	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:

$\rightsquigarrow : Set \to Set \to Set$

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

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

$o : Set$

$El\ o \hookrightarrow Prop$

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:

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

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

dependent types:

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

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

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:

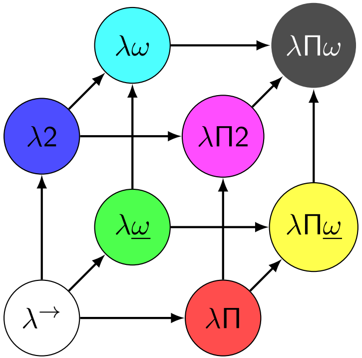
$\pi : \Pi p : Prop, (Prf\; p \to Set) \to Set$

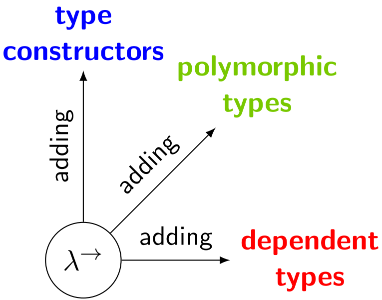
$El(\pi\; p\; a) \hookrightarrow \Pi x : Prf\; p, El(a\,x)$

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

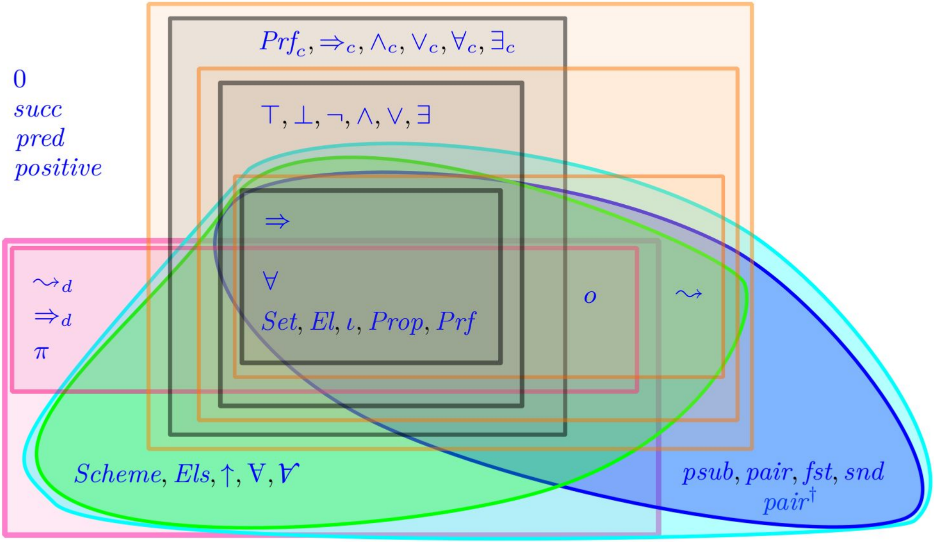
system	**PTS rule**	$\lambda\Pi/\mathcal{R}$ **rule**
simple types	TYPE, TYPE	$\mathit{Prf}\,(a \Rightarrow_d b) \hookrightarrow \Pi x : \mathit{Prf}\ a, \mathit{Prf}\,(b\,x)$
polymorphic types	KIND, TYPE	$\mathit{Prf}\,(\forall a b) \hookrightarrow \Pi x : \mathit{El}\ a, \mathit{Prf}\,(b\,x)$
dependent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





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

$El_s : U_s \to$ 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 (use recursors?)
- 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 systems 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 to Coq **scripts** using some intermediate stack-based machine language
- **Wiedijk 2007:** describes a manual translation of HOL-Light proofs in Coq terms via a **shallow embedding** (no implem)
- **Keller & Werner 2010:** translates HOL-Light proofs to Coq terms via a **deep embedding** & computational reflection

Previous works & tools on HOL to Coq

- **Denney 2000:** translates HOL98 proofs to Coq **scripts** using some intermediate stack-based machine language
- **Wiedijk 2007:** describes a manual translation of HOL-Light proofs in Coq terms via a **shallow embedding** (no implem)
- **Keller & Werner 2010:** translates HOL-Light proofs to Coq terms via a **deep embedding** & computational reflection
- **B. 2023:** implements Wiedijk approach via a **shallow embedding** in Lambdapi using results and ideas from:
  - Assaf & Burel (translation of OpenTheory to Dedukti, 2015)
  - Kaliszyk & Krauss (translation of HOL-Light to Isabelle, 2013)

HOL-Light logic

Terms: simply typed λ-terms with prenex polymorphism (OCaml)

Rules:

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

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

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

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

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

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

HOL-Light logic: connectives are defined from equality!

(Andrews Q0 logic)

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

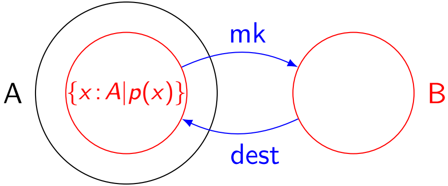
Term and type definitions in HOL-Light

- One can give a name `c` to a term `t` of type `A` by adding:
  - a typed constant `c:A`
  - an axiom `c = t`

Term and type definitions in HOL-Light

- One can give a name `c` to a term `t` of type `A` by adding:
  - a typed constant `c:A`
  - an axiom `c = t`

- One can give a name `B` to a type isomorphic to the set of terms of type `A` satisfying some predicate `p:A->bool` by adding:
  - a type constant `B`
  - a proof of ∃`a.p a`
  - a typed constant `mk:A->B`
  - a typed constant `dest:B->A`
  - an axiom ∀`b:B.mk(dest b) = b`
  - an axiom ∀`a:A.p a = (dest(mk a) = a)`



Step 1: extract proofs out of HOL-Light

HOL-Light uses the **LCF approach**:

it records provability and not proofs

```
type thm = Sequent of (term list * term      )
```

```
val REFL : term -> thm
val TRANS : thm -> thm -> thm
val MK_COMB : thm * thm -> thm
val ABS : term -> thm -> thm
val BETA : term -> thm
val ASSUME : term -> thm
val EQ_MP : thm -> thm -> thm
val DEDUCT_ANTISYM_RULE : thm -> thm -> thm
val INST_TYPE : (hol_type * hol_type) list -> thm -> thm
val INST : (term * term) list -> thm -> thm
```

Step 1: extract proofs out of HOL-Light

HOL-Light uses the **LCF approach**:

it records provability and not proofs

we need to patch it to export proofs (Obua 2005, Polu 2019):

```
type thm = Sequent of (term list * term * int)
                                        (* theorem identifier *)
val REFL : term -> thm
val TRANS : thm -> thm -> thm
val MK_COMB : thm * thm -> thm
val ABS : term -> thm -> thm
val BETA : term -> thm
val ASSUME : term -> thm
val EQ_MP : thm -> thm -> thm
val DEDUCT_ANTISYM_RULE : thm -> thm -> thm
val INST_TYPE : (hol_type * hol_type) list -> thm -> thm
val INST : (term * term) list -> thm -> thm
```

```
type proof = Proof of (thm * proof_content)

and proof_content =
| Prefl of term
| Ptrans of int * int
| ...
```

Base HOL-Light library: hol.ml

```
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
```

Step 2: simplify HOL-Light proofs

the number of generated proof steps can be reduced as follows:

- **instrument** connectives intro/elim rules & α-equivalence (20%!)

Step 2: simplify HOL-Light proofs

the number of generated proof steps can be reduced as follows:

- **instrument** connectives intro/elim rules & α-equivalence (20%!)
- **rewrite** proofs:

$$
\begin{array}{rcl}
\mathsf{SYM(REFL(t))} & \hookrightarrow & \mathsf{REFL(t)} \\
\mathsf{SYM(SYM(p))} & \hookrightarrow & \mathsf{p} \\
\mathsf{TRANS(REFL(t),p)} & \hookrightarrow & \mathsf{p} \\
\mathsf{TRANS(p,REFL(t))} & \hookrightarrow & \mathsf{p} \\
\mathsf{CONJUNCT1(CONJ(p,\_))} & \hookrightarrow & \mathsf{p} \\
\mathsf{CONJUNCT2(CONJ(\_,p))} & \hookrightarrow & \mathsf{p} \\
\mathsf{MKCOMB(REFL(t),REFL(u))} & \hookrightarrow & \mathsf{REFL(t(u))} \\
\mathsf{EQMP(REFL(\_),p)} & \hookrightarrow & \mathsf{p}
\end{array}
$$

Step 2: simplify HOL-Light proofs

the number of generated proof steps can be reduced as follows:

- **instrument** connectives intro/elim rules & α-equivalence (20%!)
- **rewrite** proofs:

$$\begin{array}{rcl}
\mathsf{SYM(REFL(t))} & \hookrightarrow & \mathsf{REFL(t)} \\
\mathsf{SYM(SYM(p))} & \hookrightarrow & \mathsf{p} \\
\mathsf{TRANS(REFL(t),p)} & \hookrightarrow & \mathsf{p} \\
\mathsf{TRANS(p,REFL(t))} & \hookrightarrow & \mathsf{p} \\
\mathsf{CONJUNCT1(CONJ(p,_))} & \hookrightarrow & \mathsf{p} \\
\mathsf{CONJUNCT2(CONJ(_,p))} & \hookrightarrow & \mathsf{p} \\
\mathsf{MKCOMB(REFL(t),REFL(u))} & \hookrightarrow & \mathsf{REFL(t(u))} \\
\mathsf{EQMP(REFL(_),p)} & \hookrightarrow & \mathsf{p}
\end{array}$$

- **remove** useless proof steps (because of tactic failures)

Step 2: simplify HOL-Light proofs

the number of generated proof steps can be reduced as follows:

- **instrument** connectives intro/elim rules & α-equivalence (20%!)
- **rewrite** proofs:

$$\begin{array}{rcl}
\mathsf{SYM(REFL(t))} & \hookrightarrow & \mathsf{REFL(t)} \\
\mathsf{SYM(SYM(p))} & \hookrightarrow & \mathsf{p} \\
\mathsf{TRANS(REFL(t),p)} & \hookrightarrow & \mathsf{p} \\
\mathsf{TRANS(p,REFL(t))} & \hookrightarrow & \mathsf{p} \\
\mathsf{CONJUNCT1(CONJ(p,_))} & \hookrightarrow & \mathsf{p} \\
\mathsf{CONJUNCT2(CONJ(_,p))} & \hookrightarrow & \mathsf{p} \\
\mathsf{MKCOMB(REFL(t),REFL(u))} & \hookrightarrow & \mathsf{REFL(t(u))} \\
\mathsf{EQMP(REFL(_),p)} & \hookrightarrow & \mathsf{p}
\end{array}$$

- **remove** useless proof steps (because of tactic failures)

initial number of steps for hol.ml	with basic tactics instrumentation	and simplification and purge
14.3 M	8.6 M (-40%)	3.5 M (-76%)

hol.ml: theory of integers, lists, real numbers, etc.

Step 3: represent HOL-Light terms and proofs in Lambdapi (Assaf & Burel, 2015)

```
/* Encoding of HOL-Light types as terms of type Set */
constant symbol Set : TYPE;
constant symbol bool : Set;
constant symbol fun : Set → Set → Set;
```

Step 3: represent HOL-Light terms and proofs in Lambdapi (Assaf & Burel, 2015)

```
/* Encoding of HOL-Light types as terms of type Set */
constant symbol Set : TYPE;
constant symbol bool : Set;
constant symbol fun : Set → Set → Set;
```

```
/* Interpretation of HOL-Light types as Lambdapi types */
injective symbol El : Set → TYPE;
rule El(fun $a $b) ↪ El $a → El $b;
```

Step 3: represent HOL-Light terms and proofs in Lambdapi (Assaf & Burel, 2015)

```
/* Encoding of HOL-Light types as terms of type Set */
constant symbol Set : TYPE;
constant symbol bool : Set;
constant symbol fun : Set → Set → Set;
```

```
/* Interpretation of HOL-Light types as Lambdapi types */
injective symbol El : Set → TYPE;
rule El(fun $a $b) ↪ El $a → El $b;
```

```
/* HOL-Light primitive constants */
constant symbol = [A] : El(fun A (fun A bool));
symbol ε [A] : El (fun (fun A bool) A);
```

Step 3: represent HOL-Light terms and proofs in Lambdapi (Assaf & Burel, 2015)

```
/* Encoding of HOL-Light types as terms of type Set */
constant symbol Set : TYPE;
constant symbol bool : Set;
constant symbol fun : Set → Set → Set;
```

```
/* Interpretation of HOL-Light types as Lambdapi types */
injective symbol El : Set → TYPE;
rule El(fun $a $b) ↪ El $a → El $b;
```

```
/* HOL-Light primitive constants */
constant symbol = [A] : El(fun A (fun A bool));
symbol ε [A] : El (fun (fun A bool) A);
```

```
/* Interpretation of HOL-Light propositions as Lambdapi types
   (Curry-Howard correspondence to be defined) */
injective symbol Prf : El bool → TYPE;
```

Step 3: represent HOL-Light terms and proofs in Lambdapi (Assaf & Burel, 2015)

```
/* HOL-Light axioms and rules */
symbol REFL [a] (t : El a) : Prf(= t t);
symbol MK_COMB [a b] [s t : El(fun a b)] [u v : El a] :
  Prf(= s t) → Prf(= u v) → Prf(= (s u) (t v));
symbol EQ_MP [p q] : Prf(= p q) → Prf p → Prf q;
symbol fun_ext [a b] [f g : El (fun a b)] :
  (Π x, Prf (= (f x) (g x))) → Prf (= f g);
symbol prop_ext [p q] :
  (Prf p → Prf q) → (Prf q → Prf p) → Prf (= p q);
```

Step 3: represent HOL-Light terms and proofs in Lambdapi (Assaf & Burel, 2015)

```
/* HOL-Light axioms and rules */
symbol REFL [a] (t : El a) : Prf(= t t);
symbol MK_COMB [a b] [s t : El(fun a b)] [u v : El a] :
  Prf(= s t) → Prf(= u v) → Prf(= (s u) (t v));
symbol EQ_MP [p q] : Prf(= p q) → Prf p → Prf q;
symbol fun_ext [a b] [f g : El (fun a b)] :
  (Π x, Prf (= (f x) (g x))) → Prf (= f g);
symbol prop_ext [p q] :
  (Prf p → Prf q) → (Prf q → Prf p) → Prf (= p q);
```

```
/* HOL-Light derived connectives */
constant symbol ⇒ : El (fun bool (fun bool bool));
rule Prf(⇒ $p $q) ↪ Prf $p → Prf $q;
constant symbol ∀ [A] : El (fun (fun A bool) bool);
rule Prf(∀ $p) ↪ Π x,Prf($p x);
...
```

Step 3: represent HOL-Light terms and proofs in Lambdapi (Assaf & Burel, 2015)

```
/* HOL-Light axioms and rules */
symbol REFL [a] (t : El a) : Prf(= t t);
symbol MK_COMB [a b] [s t : El(fun a b)] [u v : El a] :
  Prf(= s t) → Prf(= u v) → Prf(= (s u) (t v));
symbol EQ_MP [p q] : Prf(= p q) → Prf p → Prf q;
symbol fun_ext [a b] [f g : El (fun a b)] :
  (Π x, Prf (= (f x) (g x))) → Prf (= f g);
symbol prop_ext [p q] :
  (Prf p → Prf q) → (Prf q → Prf p) → Prf (= p q);
```

```
/* HOL-Light derived connectives */
constant symbol ⇒ : El (fun bool (fun bool bool));
rule Prf(⇒ $p $q) ↪ Prf $p → Prf $q;
constant symbol ∀ [A] : El (fun (fun A bool) bool);
rule Prf(∀ $p) ↪ Π x,Prf($p x);
...
```

```
/* Natural deduction rules */
symbol ∧i [p] : Prf p → Π[q],Prf q → Prf(∧ p q);
symbol ∧e1 [p q] : Prf(∧ p q) → Prf p;
symbol ∧e2 [p q] : Prf(∧ p q) → Prf q;
symbol ∃i [a] (p : El a → El bool) t : Prf(p t) → Prf(∃ p);
symbol ∃e [a] [p : El a → El bool] :
  Prf(∃(λ x,p x)) → Π[r],(Π x:El a,Prf(p x) → Prf r) → Prf r;
```

Step 4: from Lambdapi to Coq

the translation is purely syntactic:

- the symbols El and Prf are removed
- some symbols are replaced by Coq expr. wrt a user-defined map:

HOL-Light	Lambdapi	Coq
hol_type	Set	{type:>Type; el:type}
fun	arr	->
bool	bool	Prop
=	=	eq
Prefl	REFL	eq_refl
==>	$\Rightarrow$	->
/\	$\wedge$	and
num	num	nat
+	+	add
<=	<=	le
...	...	...

example output:

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

Step 5: alignment of definitions

- One can give a name `c` to a term `t` of type `A` by adding:
  - a typed constant `c:A`
  - an axiom `c = t`

Step 5: alignment of definitions

- One can give a name `c` to a term `t` of type `A` by adding:
  - a typed constant `c:A`
  - an axiom `c = t`

  to replace `c` by the Coq expression `c'`, we need to do in Coq:
  - prove $\boxed{\texttt{c' = t}}$

Step 5: alignment of definitions

- One can give a name `c` to a term `t` of type `A` by adding:
  - a typed constant `c:A`
  - an axiom `c = t`

  to replace `c` by the Coq expression `c'`, we need to do in Coq:
  - prove $\boxed{\texttt{c' = t}}$

- One can give a name `B` to a type isomorphic to the set of terms of type `A` satisfying some predicate `p:A->bool` by adding:
  - a type constant `B`
  - a proof of $\exists$`a.p a`
  - a typed constant `mk:A->B`
  - a typed constant `dest:B->A`
  - an axiom $\forall$`b:B.mk(dest b) = b`
  - an axiom $\forall$`a:A.p a = (dest(mk a) = a)`

  to replace `B` by the Coq expression `B'`, we need to do in Coq:
  - define $\boxed{\texttt{mk:A->B'}}$
  - define $\boxed{\texttt{dest:B'->A}}$
  - prove $\boxed{\forall\texttt{b:B', mk(dest b) = b}}$
  - prove $\boxed{\forall\texttt{a:A, p a = (dest(mk a) = a)}}$

Alignments already proved

- **connectives**
- **unit** type
- **product** type constructor
- type of **natural numbers**, addition, substraction, multiplication, division, power, ordering, min, max, mod, even, odd, ...
- **option** type constructor
- **sum** type constructor
- **list** type constructor, head, tail, concatenation, reverse, length, map, forall, membership, ... (thanks to Anthony Bordg)

and we are currently working on the type of **real** numbers

HOL-Light library in Coq

available on Opam:

https://github.com/deducteam/coq-hol-light/

currently contains 667 lemmas on logic, arithmetic and lists mainly

usage in Coq:

```
Require Import HOLLight.hol_light.
```

Axioms required in Coq

```
Axiom classic (P : Prop) : P \/ ~ P.

Axiom constructive_indefinite_description (A : Type) P :
  (exists x, P x) -> {x : A | P x}.

Axiom fun_ext {A B: Type} {f g: A -> B}:
  (forall x, f x = g x) -> f = g.

Axiom prop_ext {P Q : Prop} : (P -> Q) -> (Q -> P) -> P = Q.

Axiom proof_irrelevance (P:Prop) (p1 p2 : P) : p1 = p2.
```

Performances

The translations (HOL-Light to Lambdapi, and Lambdapi to Coq) and the verification by Coq can be done **in parallel** by generating a Lambdapi/Coq file for each HOL-Light user-defined theorem

To scale up, we also need to **share** types and terms

On a machine with 32 processors i9-13950HX and 64Gb RAM:

HOL-Light file	dump-simp	dump size	proof steps	nb theorems
hol.ml	3m57s	3 Gb	5 M	5679
topology.ml	48m	52 Gb	52 M	18866

HOL-Light file	make -j32 lp	make -j32 v	v files size	make -j32 vo
hol.ml	51s	55s	1 Gb	18m4s
topology.ml	22m22s	20m16s	68 Gb	8h

Tools: hol2dk and 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

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!