Proof Systems Interoperability

Frédéric Blanqui



Outline

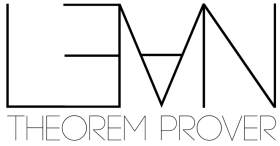
Historical overview on proof systems interoperability

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

Example: from HOL-Light to Rocq via Lambdapi

Proof systems







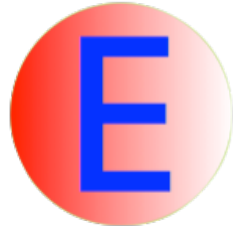




HOL
Interactive Theorem Prover

















. . .

Libraries of formal proofs today

(data on May 2026)

library	files	theorems
Rocq Opam	54,300	748,000
Isabelle AFP	10,000	315,000
Lean Mathlib	8,500	273,000
HOL	...	...
Mizar Mathlib	1,500	45,000
HOL-Light	670	41,000
PVS	...	...
Agda	...	...
...	...	...

- 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
- Help changing 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 in a system not supporting that feature)

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 termination of rewrite systems
  TSTP proof format for ATPs
  SMT-lib format for FOL modulo theory problems
  Flyspeck project with HOL-Light, Rocq 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]

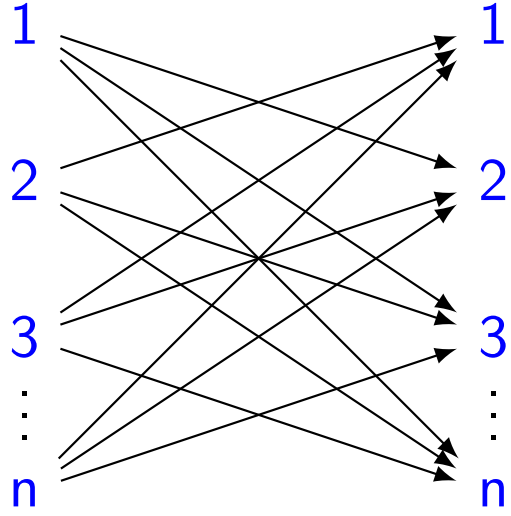
One-to-one translation tools

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

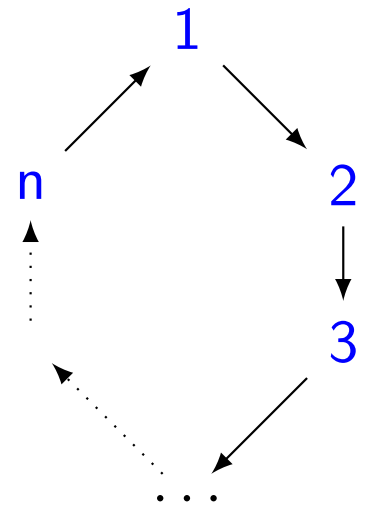
Flyspeck project with HOL-Light, Rocq and Isabelle/HOL [2003]

- HOL to Isabelle/HOL [Obua 2006]
- Isabelle/HOL to HOL-Light [McLaughlin 2006]
- HOL-Light to Rocq [Wiedijk 2007, no implementation]
- HOL-Light to Rocq [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 Rocq [Gilbert 2020]
- Lean3 to Lean4 [Lean community 2021]
- Maude to Lean [Rubio & Riesco 2022]
- ...

Interoperability between n systems ?

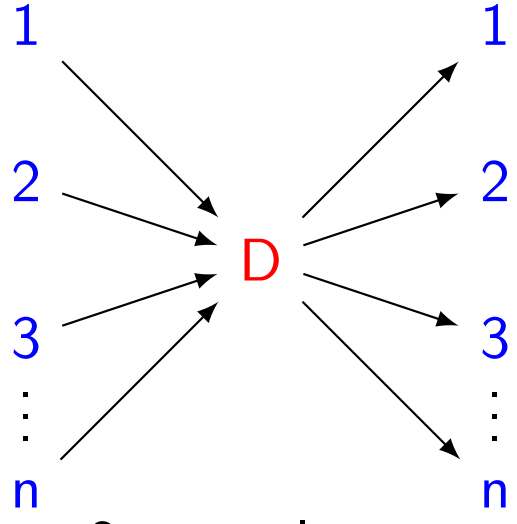


$n(n-1)$ translators



n translators

Can't we be more generic ?



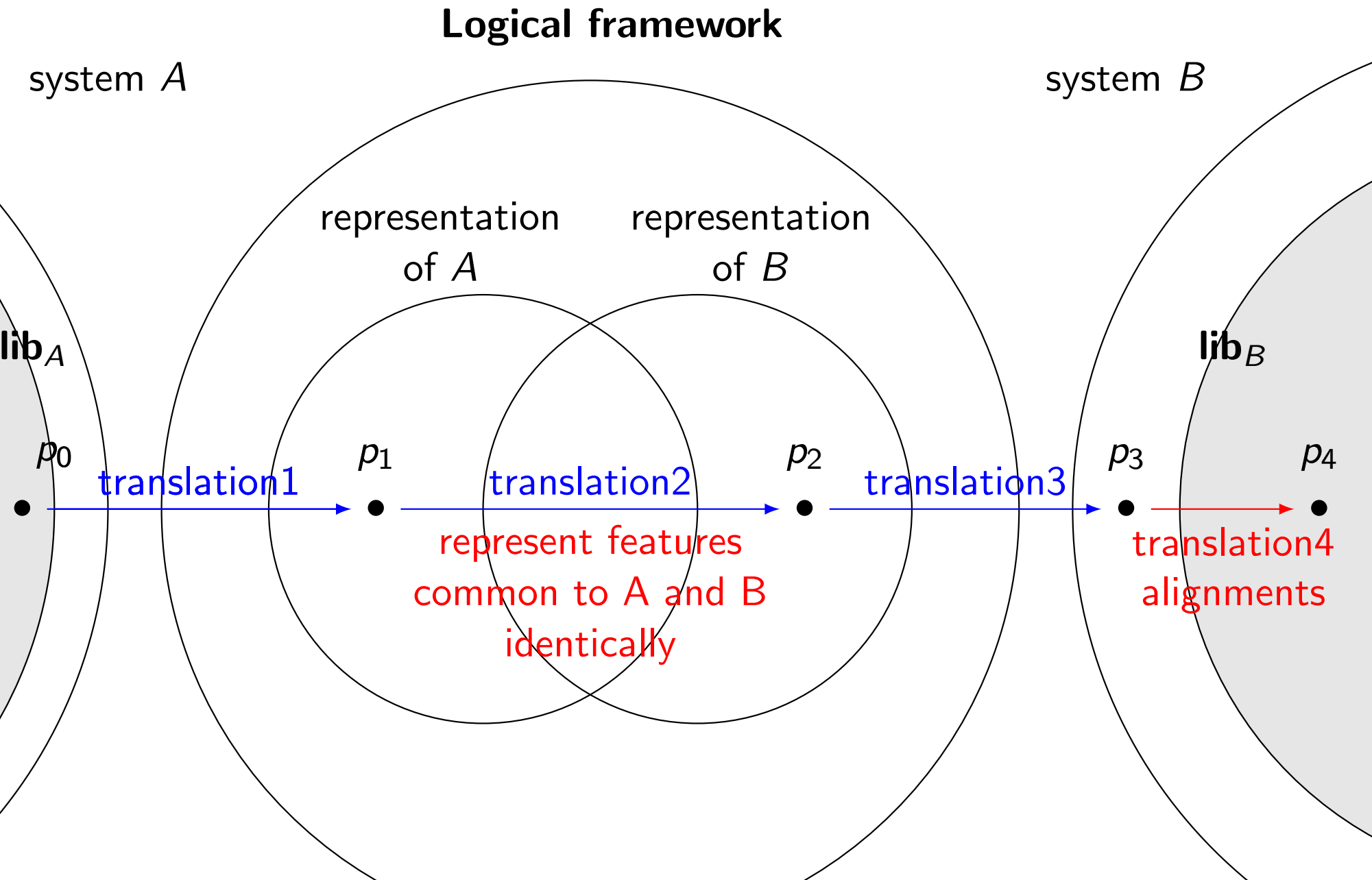
$2n$ translators

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

Deducteam approach to proof systems interoperability



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

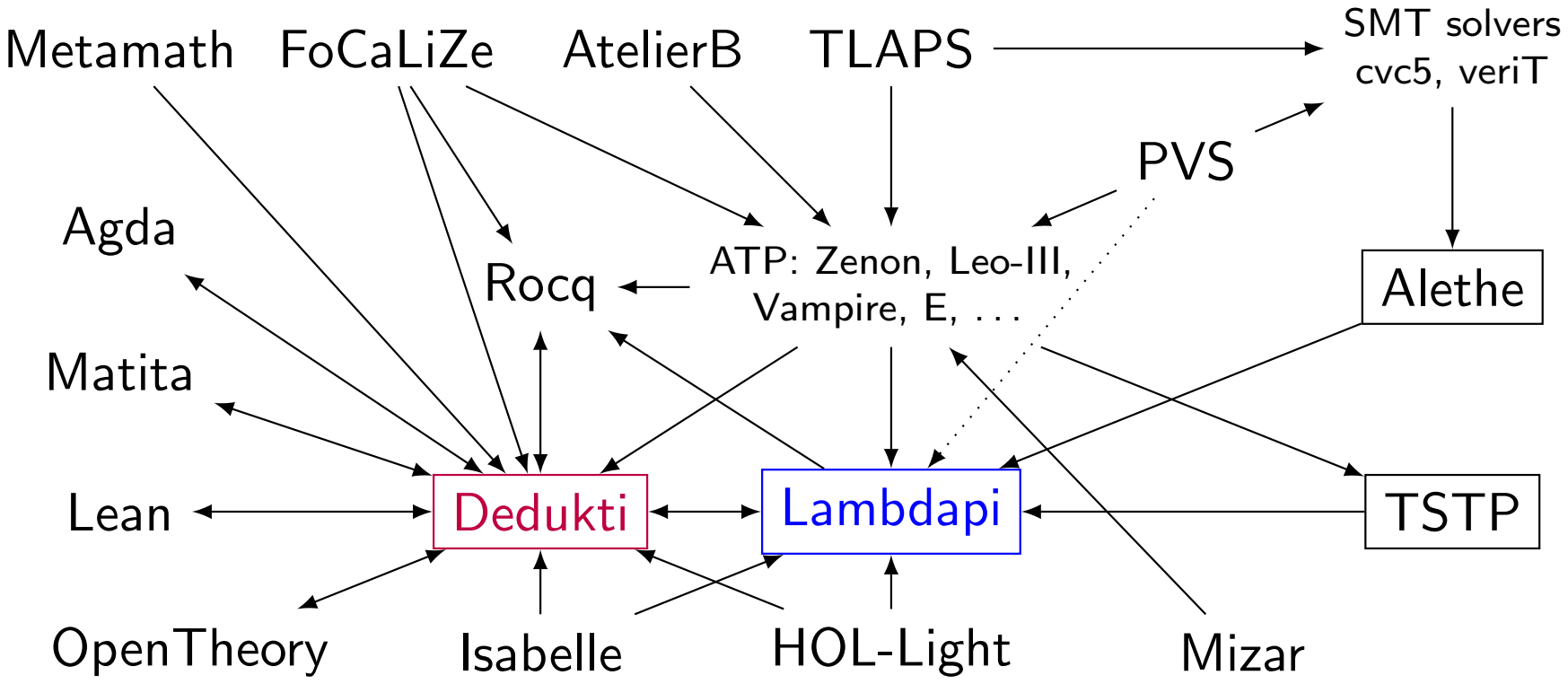
allows one to represent S=geometry, S=arithmetic, S=set theory, ...
not well suited for computation and dependent types

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

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

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

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
Lean	stdlib, parts of mathlib 
HOL-Light	stdlib, `Logic`, `Multivariate`
OpenTheory	OpenTheory library
Matita	arithmetic library
Rocq	stdlib parts, GeoCoq parts
Isabelle	HOL session
Agda	stdlib parts ($\pm$ 25%)
PVS	stdlib parts (statements only)
TPTP	E 69%, Vampire 83% (for CNF only), Leo-III  integration in TPTP World via GDV

Thanks to Claudio Sacerdoti, Dedukti libraries can be searched by using the Lambdapi index and search commands and online on:

https://lambdapi.saclay.inria.fr/

Examples of translations via Dedukti

- HOL-Light $\longrightarrow$ Rocq [B. 2024, Gontard 2025]
  https://github.com/Deducteam/rocq-hollight
- Matita arith lib $\longrightarrow$ Agda [Felicissimo 2023]
  https://github.com/thiagofelicissimo/matita_lib_in_agda
- Matita arith lib $\longrightarrow$ OpenTheory, Rocq, PVS, Lean [Thiré 2018]
  http://logipedia.inria.fr

Outline

Historical overview on proof systems interoperability

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

Example: from HOL-Light to Rocq via Lambdapi

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

concat 2 a 3 b : Array$(2+3) \equiv_{\beta\mathcal{R}}$ 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** $I : \texttt{TYPE}$
  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: $I \to \ldots \to I \to Prop$
  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 $\forall(\lambda x : I, A)$
  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

but

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

and

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

unless we add the rewrite rule

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

Encoding $\Rightarrow$

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

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

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

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

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

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

Encoding $\forall$

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

$$\mathit{Prf}\,(\forall p) \quad \hookrightarrow \quad \Pi x : I, \mathit{Prf}\,(p\,x)$$

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:

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

introduction and elimination rules can be derived:

($\wedge$-intro):

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

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

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

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

$a : Prf\ A$ for each axiom A

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

$$Prf\,(A \Rightarrow B) \hookrightarrow Prf\ A \to Prf\ B$$
$$Prf\,(\forall A) \hookrightarrow \Pi x : I, Prf\,(A\,x)$$
$$Prf\,(A \wedge B) \hookrightarrow \Pi C : Prop, (Prf\ A \to Prf\ B \to Prf\ C) \to Prf\ C$$
$$Prf\,\bot \hookrightarrow \Pi C : Prop, Prf\ C$$
$$Prf\,(\neg A) \hookrightarrow Prf\,A \to Prf\,\bot$$

$\ldots$

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

encoding of terms:

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

encoding of contexts:

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

encoding of propositions:

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

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 ι
- 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 translated back to first-order logic and natural deduction easily

Multi-sorted first-order logic

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

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

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

Polymorphic first-order logic

same trick as for the BHK interpretation of propositions:

$\mathit{Set} : \mathtt{TYPE}$ — type of sorts

$\mathit{El} : \mathit{Set} \to \mathtt{TYPE}$ — interpretation of sorts as types

$\iota : \mathit{Set}$ — for each sort ι

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

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

Higher-order logic

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

quantification on functions:

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

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

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:

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

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

dependent types:

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

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

proofs in object-terms:

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

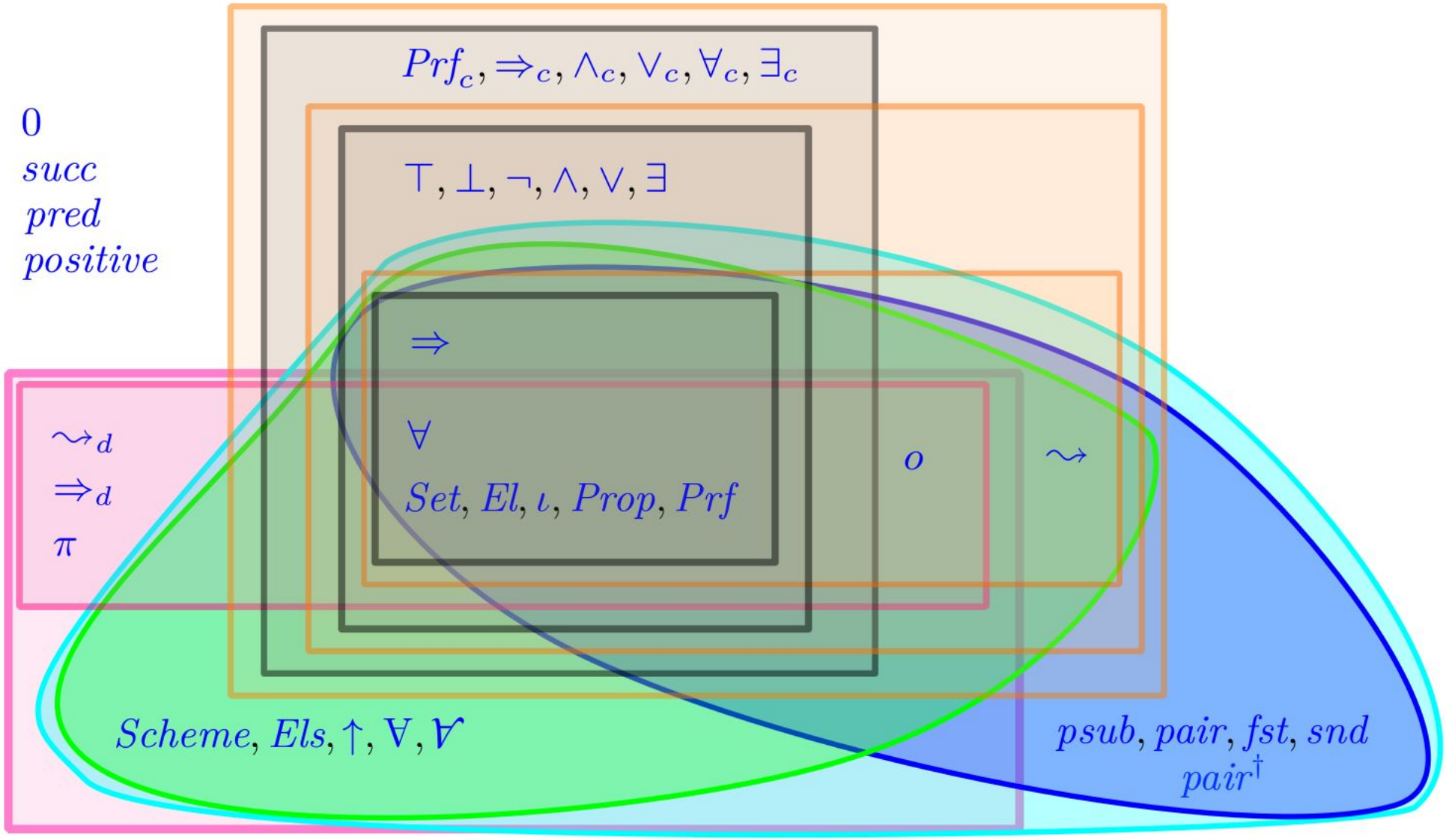
$\mathit{El}(\pi\ p\ a) \hookrightarrow \Pi x : \mathit{Prf}\ p, \mathit{El}(a\,x)$

example: $\mathit{div} : \mathit{El}(\iota \rightsquigarrow \iota \rightsquigarrow_d \lambda y : \mathit{El}\ \iota, \pi(y > 0)(\lambda_, \iota))$
$\equiv_{\beta\mathcal{R}} \Pi x : \mathit{El}\ \iota, \Pi y : \mathit{El}\ \iota, \Pi p : \mathit{Prf}\,(y > 0), \mathit{El}\ \iota$

takes 3 arguments: $x : \mathit{El}\ \iota$, $y : \mathit{El}\ \iota$, $p : \mathit{Prf}\,(y > 0)$
and returns a term of type $\mathit{El}\ \iota$

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

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



Lambdapi files

Encoding other features

- recursive functions [Assaf 2015, Cauderlier 2016, Férey 2021]
  - different approaches, no common representation yet (use recursors?)
- universe cumulativity [Assaf 2015, Thiré 2020, Férey 2021]
- η-conversion on function types [Genestier 2020]
- predicative universe polymorphism [Genestier 2020]
  - requires rewriting with matching modulo AC
    or rewriting on AC canonical forms [B. 2022]
- predicate subtyping with proof irrelevance [Hondet 2020]
- co-inductive objects and co-recursion [Felicissimo 2021]
- impredicative universe polymorphism [Géran 2026]

Outline

Historical overview on proof systems interoperability

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

Example: from HOL-Light to Rocq via Lambdapi

HOL-Light Multivariate library

A library with many results in analysis

```
loadt "Multivariate/metric.ml";;        (* General topology, metric spac
loadt "Multivariate/homology.ml";;      (* Singular homology theory
loadt "Multivariate/vectors.ml";;       (* Basic vectors, linear algebra
loadt "Multivariate/determinants.ml";; (* Determinant and trace
loadt "Multivariate/topology.ml";;      (* Topology of R^n and much else
loadt "Multivariate/convex.ml";;        (* Convex sets and functions
loadt "Multivariate/paths.ml";;         (* Paths, simple connectedness e
loadt "Multivariate/polytope.ml";;      (* Faces, polytopes, polyhedra e
loadt "Multivariate/degree.ml";;        (* Brouwer degree, retracts etc.
loadt "Multivariate/derivatives.ml";;  (* Derivatives
loadt "Multivariate/clifford.ml";;      (* Geometric (Clifford) algebra
loadt "Multivariate/integration.ml";;  (* Integration, bounded variatio
loadt "Multivariate/measure.ml";;       (* Lebesgue measure
loadt "Multivariate/complexes.ml";;        (* Complex numbers
loadt "Multivariate/canal.ml";;            (* Complex analysis
loadt "Multivariate/transcendentals.ml";; (* Real & complex transcenden
loadt "Multivariate/realanalysis.ml";;     (* Some analytical stuff on R
loadt "Multivariate/moretop.ml";;          (* Further topological result
loadt "Multivariate/cauchy.ml";;           (* Complex line integrals
```

HOL-Light Logic library

A library with many metatheoretical results in first-order logic

```
loadt "Logic/fol.ml";;          (* Basic first order logic definitions
loadt "Logic/prenex.ml";;       (* Prenex normal form
loadt "Logic/skolem.ml";;       (* Skolem normal form
loadt "Logic/fol_prop.ml";;     (* Compactness etc. for propositional cas
loadt "Logic/canon.ml";;        (* First order case via canonical models
loadt "Logic/herbrand.ml";;     (* Refinement to ground terms
loadt "Logic/fole.ml";;         (* First order logic with equality
loadt "Logic/unif.ml";;         (* Unification algorithms and MGUs
loadt "Logic/resolution.ml";;   (* Completeness of resolution
loadt "Logic/given.ml";;        (* More refined analysis of subsumption e
loadt "Logic/linear.ml";;       (* Linear refinements
loadt "Logic/support.ml";;      (* Set-of-support tautology-free versions
loadt "Logic/positive.ml";;     (* Positive and general semantic resoluti
loadt "Logic/givensem.ml";;     (* Adaptation of given.ml to semantic cas
loadt "Logic/prolog.ml";;       (* Backchaining for Horn clauses
loadt "Logic/birkhoff.ml";;     (* Birkhoff's completeness theorem
loadt "Logic/trs.ml";;          (* Results on simplifying convergent TRS
loadt "Logic/lpo.ml";;          (* The lexicographic path ordering
```

Previous works & tools on HOL to Rocq

- **Denney 2000:** translates HOL98 proofs to Rocq **scripts** using some intermediate stack-based machine language
- **Wiedijk 2007:** describes a manual translation of HOL-Light proofs in Rocq terms via a **shallow embedding** (no implem)
- **Keller & Werner 2010:** translates HOL-Light proofs to Rocq 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!

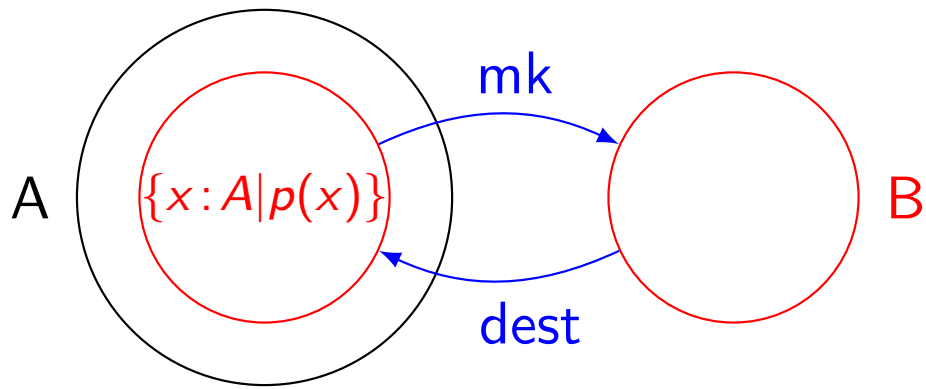
(Andrews Q0 logic)

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

Term and type definitions in HOL-Light

- One can give a name `c` to a term `t: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
| ...
```

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:

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

- **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;
```

```
/* 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);
```

```
/* 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 Rocq

The translation is easy, it essentially consists in:

- renaming keywords
- removing the symbols `El` and `Prf` used for the encoding

Example:

- **Lambdapi:**

```
symbol fun_ext [a b] [f g : El (fun a b)] :
  (Π x, Prf (= (f x) (g x))) → Prf (= f g);
```

- **Rocq:**

```
Axiom fun_ext {a b} {f g : a -> b} :
  (forall x, (= (f x) (g x))) -> (= f g);
```

But translation is not enough . . .

Rocq users want to use HOL-Light theorems *on* Rocq types/functions

$\Rightarrow$ we need to replace in translated statements
HOL-Light types/functions by their Rocq counterparts

e.g. the HOL-Light type `num` for natural numbers should be replaced by the Rocq type `nat`

But when is such an *alignment* valid?

Step 5: alignment of term definitions

Reminder: in HOL-Light,
one can give a name `c` to a term `t:A` by adding:

- a typed constant `c:A`
- an axiom `c = t`

Then, any proof about `c` only relies on that axiom

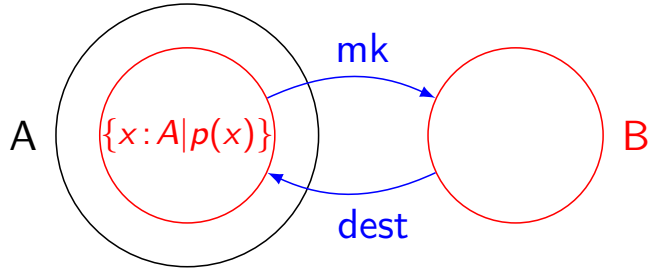
$\Rightarrow$ we can safely replace `c` by `c'` as long as we can:

- prove $\boxed{\texttt{c' = t}}$

Step 5: alignment of type definitions

Reminder: in HOL-Light,
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)`



then, any proof about `B` only relies on those constants and axioms

⇒ we can safely replace `B` by `B'` as long as we can:

- define `mk:A->B'`
- define `dest:B'->A`
- prove `∀b:B', mk(dest b) = b`
- prove `∀a:A, (p a) = (dest(mk a) = a)`

Default alignment of HOL-Light types

```
Section Subtype.

 Variables (A: Type) (P: A->Prop) (a: A) (h: P a).

 Definition subtype :=
   {| type := {x:A|P x}; el := exist P a h |}.

 Definition dest : subtype->A := fun x => proj1_sig x.

 Definition mk : A->subtype := fun x =>
  COND_dep (P x) subtype (exist P x) (fun _ => exist P a h).

 Lemma dest_mk x : P x = (dest (mk x) = x).
 Proof. ... (*omitted*) Qed.

 Lemma mk_dest x : mk (dest x) = x.
 Proof. ... (*omitted*) Qed.

End Subtype.
```

Inductive predicates in HOL-Light and Rocq

Example: finiteness of a subset `S:A->Prop` of `A`

- `EMPTY A := False`
- `INSERT A x S := fun y => y = x ∨ S y`

- **HOL-Light:** smallest predicate satisfying some conditions

```
Definition FINITE (A: Type) (S: A->Prop): Prop :=
  forall F: (A->Prop)->Prop,
    (forall T:A->Prop,
      (T = EMPTY A
      \/ exists x U, T = INSERT A x U /\ F U)
      -> F T)
    -> F S
```

- **Rocq:** inductive predicates are primitive

```
Inductive Finite (A: Type): (A->Prop)->Prop :=
| EMPTY_is_finite: Finite (EMPTY A)
| INSERT_is_finite:
  forall x S, Finite S -> Finite (INSERT A x S).
```

their equality can be proved automatically in Rocq by using the tactic `ind_align` [Gontard 2025] 

Inductive types in HOL-Light and Rocq

- **HOL-Light:** an inductive type is defined as being isomorphic to a subset of the tree type `recspace` α whose constructors are:
  - `BOTTOM: recspace` α
  - `CONSTR: num -> ` α `-> (num -> recspace` α`) -> recspace` α

  Example: `list` $\alpha \underset{\texttt{mk}}{\overset{\texttt{dest}}{\rightleftharpoons}}$ `recspace` α
  - `nil` $=$ `mk(CONSTR 0` *d* `FNIL)`
  - `cons` $= \lambda$*a* $: \alpha, \lambda$*l* $:$ `list` $\alpha,$ `mk(CONSTR 1` *a* `(FCONS (dest` *l*`) FNIL))`
- **Rocq:** inductive types are primitive

```
Inductive list (A : Type) :=
  nil : list A | cons : A -> list A -> list A.
```

their equality can be proved automatically in Rocq by using the tactics `mk_dest` and `dest_mk` [Gontard 2025] 

Recursive functions in HOL-Light and Rocq

- **HOL-Light:** a recursive function is defined using the choice operator ε as some function satisfying some equations

  For example, the addition on naturals is *internally* defined as:

```
ε (fun add':nat->nat->nat->nat, forall a:nat,
     (forall n:nat, add' a (NUMERAL 0) n = n
  /\ (forall m:nat, forall n:nat,
                    (add' a (S m) n) = (S (add' a m n)))))
  (NUMERAL (BIT1 (BIT1 (BIT0 (BIT1 (BIT0 (BIT1 0)))))))
```

- **Rocq:** recursive functions are defined using pattern-matching

```
Fixpoint add n m :=
  match n with 0 => m | S p => S (add p m) end.
```

their equality can be proved automatically in Rocq by using the tactic `total_align` [Gontard 2025] 

Rocq tactics for automating alignments (Gontard 2025)

- **ind_align**: prove the equality of a Rocq predicate P defined with `Inductive` and its HOL-Light definition as the smallest predicate P such that ...
- **mk_dest**: prove equalities of the form $mk(dest\ x) = x$ when $mk = dest^{-1}$ and $f^{-1} = \lambda y, \varepsilon(\lambda x, y = fx)$
- **dest_mk**: prove equalities of the form $P\,x = (dest(mk\,x) = x)$ when $mk = dest^{-1}$ and $f^{-1} = \lambda y, \varepsilon(\lambda x, y = fx)$
- **total_align**: prove the equality of a Rocq function defined with `Fixpoint` and its HOL-Light ε-definition

Alignment of real numbers

1. HOL-Light first defines the type `nadd` as being isomorphic to the subset of the functions $f : \texttt{num} \to \texttt{num}$ that are nearly additive, *i.e.* $\exists B, \forall m, \forall n, |m \times f(n) - n \times f(m)| \leq B \times (m + n)$.
2. It then defines the type `hreal` of non-negative reals as being isomorphic to $\texttt{nadd}/R$ where $R\,x\,y = \exists B, \forall n, |x_n - y_n| \leq B$ (quotients are a particular case of subset).
3. Finally, it defines `real` as isomorphic to $(\texttt{hreal} \times \texttt{hreal})/R$ where $R\,(x_1, y_1)\,(x_2, y_2) = (x_1 +_{\texttt{hreal}} y_2 = x_2 +_{\texttt{hreal}} y_1)$.

this definition of real numbers can be aligned with any other definition as the **theory of Dedekind-complete ordered fields is categorical** (any two models are isomorphic) [Huntington, 1903]

done for the Rocq stdlib and Mathcomp using the proof of Huntington's theorem in the Fourcolor lib [Gonthier 2006]

Alignments currently proved

- **propositions:** connectives and their intro/elim rules, equality, ε
- **relations:** well-foundedness and accessibility predicates
- **unit type:** its unique constructor
- **sum type:** its constructors
- **option type:** its constructors
- **product type:** constructor and projections
- **natural numbers:** 0, successor, doubling function, predecessor, $+$, $\times$, exponentiation, $\leq$, $<$, $\geq$, $>$, min, max, factorial, Euclidian division $/$, modulo, even and odd predicates
- **lists:** constructors, concatenation, reversal, length, map, map2, last element removal, forall/exists predicates, filter, membership, element repetition, iterator, head, tail, last element, n-th element, zip, fold, ...
- **reals:** $\leq$, $<$, $\geq$, $>$, $+$, $-$, $\times$, $/$, inverse $.^{-1}$, opposite, exponentiation, absolute value $|\ |$, max, min, sign, square root $\sqrt{.}$, modulo
- **integers:** injection of natural numbers, $\leq$, $<$, $\geq$, $>$, $+$, $-$, $|\ |$, sign, max, min, exponentiation, Euclidian division, modulo, remainder, co-primality, gcd, lcm
- **sets:** $\emptyset$, $\in$, $\cup$, $\cap$, image, difference, singleton, disjointness
- **finite sets:** finiteness predicate, cardinal, iterator, conversion to/from lists
- **analysis:** $\mathbb{R}^n$ vector spaces, operations on vectors, metric spaces, limit, ... 

HOL-Light library in Rocq

available on Opam:

https://github.com/deducteam/rocq-hollight/

contains more than 20,000 lemmas on logic, arithmetic, real and complex analysis, topology, ...

usage in Rocq:

```
From HOLLight Require Import Multivariate.theorems.
```

Output examples

```
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).
```

```
Lemma thm_SERIES_RATIO : forall {N' : Type'} (c : R)
  (a : nat -> cart R N') (s : nat -> Prop) (N'' : nat), c < 1
  /\ (forall n : nat, (N'' <= n)%N ->
      vector_norm (a n.+1) <= c * vector_norm (a n))
  -> exists l : cart R N',
     \sum_(i < n.+1 | s i) a i @[n --> \oo] --> l.
```

Axioms used in Rocq

```
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.
```

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
  - converts dk/lp files in some HOL theory into Rocq 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]
- some progress on complex structures is ongoing, e.g. translation
  of type classes between Isabelle & Rocq [Gontard]