# Encoding type universes without using matching modulo associativity and commutativity

Frédéric Blanqui

Deduc⊢eam

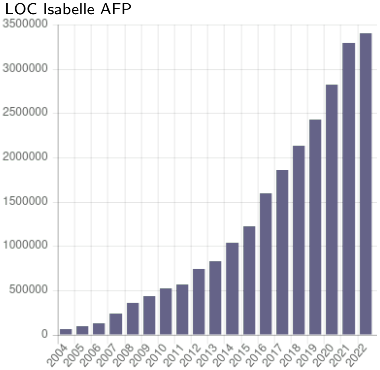
EuroProofNet

4 August 2022

# Libraries of formal proofs today

| Library | Nb files | Nb objects* |
|---|---|---|
| Coq Opam | 16,000 | 473,000 |
| Isabelle AFP | 7,000 | 90,000 |
| Lean Mathlib | 2,000 | 81,000 |
| Mizar Mathlib | 1,400 | 77,000 |
| HOL-Light | 500 | 35,000 |
| ... | ... | ... |

* type, definition, theorem, ...




- Every system has the same basic libraries on arithmetic, lists, ...
- Some definitions/theorems are available in one system only (odd-order theorem, compcert-C, seL4, perfectoid space, ...)

# Can we translate proofs from one system to the other ?

Why?

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

# Can we translate proofs from one system to the other ?

Why?

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

Problems:

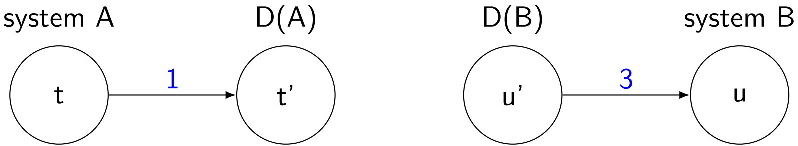
- Each system is based on different axioms and deduction rules
- It is generally non trivial and sometimes impossible to translate a proof from one system to the other
  (e.g. a classical proof in an intuitionistic system)

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

0. take a logical framework $D$ in which you can encode $A$ and $B$ so that features common to $A$ and $B$ are encoded identically

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

0. take a logical framework $D$ in which you can encode $A$ and $B$ so that features common to $A$ and $B$ are encoded identically

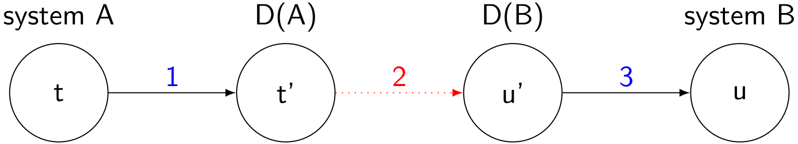



1. translate $t \in A$ to $t' \in D(A)$

3. translate $u' \in D(B)$ to $u \in B$

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

0. take a logical framework $D$ in which you can encode $A$ and $B$ so that features common to $A$ and $B$ are encoded identically




1. translate $t \in A$ to $t' \in D(A)$
2. translate $t' \in D(A)$ to $u' \in D(B)$ (if possible)
3. translate $u' \in D(B)$ to $u \in B$

# Example of logical framework: the $\lambda\Pi$-calculus modulo rewriting ($\lambda\Pi/\mathcal{R}$)

| | |
|---|---|
| $\lambda$ | simply-typed $\lambda$-calculus |
| $\Pi$ | dependent types, e.g. $B = \mathit{array}(x)$ for arrays of size $x$ |
| $\mathcal{R}$ | identification of types modulo a *convergent* rewrite system $\mathcal{R}$ |

**terms**

| | |
|---|---|
| $\star$ | sort of types |
| $f$ | global constant |
| $x$ | local variable |
| $tu$ | application |
| $\lambda x : t, u$ | abstraction |
| $\Pi x : t, u$ | dependent product |
| $t \to u$ | abbrev. for $\Pi x : t, u$ when $x \notin u$ |

**types**

$$\frac{\Gamma, x : A \vdash t : B \quad \Gamma \vdash \Pi x : A, B : \star}{\Gamma \vdash \lambda x : A, t : \Pi x : A, B}$$

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

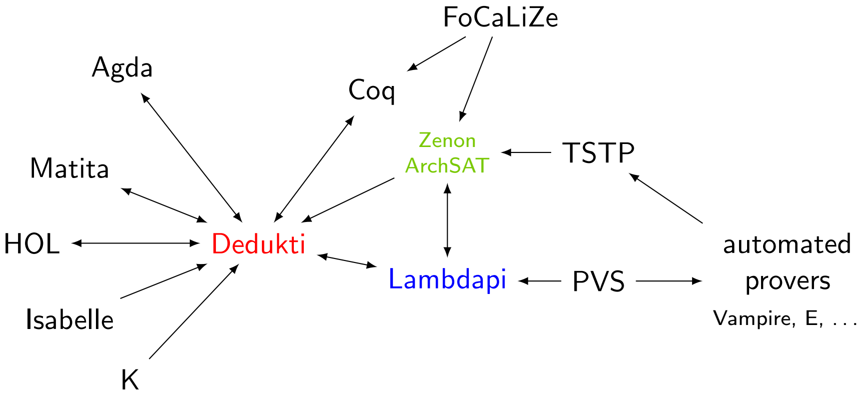
$$\frac{\Gamma \vdash t : A \quad A \equiv_{\beta\mathcal{R}} B}{\Gamma \vdash t : B} \quad \ldots$$

$$\mathit{concat} : \Pi x : \mathbb{N}, \mathit{array}(x), \Pi y : \mathbb{N}, \mathit{array}(y) \to \mathit{array}(x + y)$$

$$\mathit{array}(2 + 3) \equiv_{\beta\mathcal{R}} \mathit{array}(5)$$

# $\lambda\Pi/\mathcal{R}$ in practice: the Dedukti language

functional Pure Type Systems [Cousineau&Dowek, 2007] but also:




Dedukti is a concrete language for $\lambda\Pi/\mathcal{R}$
Lambdapi is a proof assistant for Dedukti

# On the origin of type theory

solutions proposed to overcome Russell's paradox in set theory:

- restrict the comprehension scheme
- use "types" to classify sets

  example: in simple type theory
  - ur elements are of type $\iota$
  - sets of ur elements are of type $\iota \to o$
  - sets of sets of ur elements are of type $(\iota \to o) \to o$
  - ...

# Universes

- a universe $U$ is a set of types closed by exponentiation

$$\frac{A \in U \quad B \in U}{A \to B \in U}$$

  example: the set $U_0$ of the simple types $\iota$, $\iota \to o$, ...

- universes are like inaccessible cardinals in set theory:
  - an inaccessible cardinal is closed by set exponentiation
  - a universe is closed by type exponentiation

# More universes

- some math. constructions quantifies over the elements of $U_0$
  $\Rightarrow$ they need to inhabit a new universe $U_1$ containing $U_0$

- by iteration we get an infinite sequence of nested universes

$$U_0 \in U_1 \in \ldots U_i \in U_{i+1} \ldots$$

$$\frac{A \in U_i \quad B \in U_j}{A \to B \in U_{\max(i,j)}}$$

available in some proof assistants like Coq, Agda, Lean

# Universe polymorphism

some proof assistants go further: fixed universe levels $0, 1, \ldots$ are replaced by *open* terms of the **max-successor algebra $\mathcal{L}$:**

$$t, u = x \in \mathcal{V} \mid \mathtt{z} \mid \mathtt{s}\, t \mid t \sqcup u$$

and universes having equivalent levels are identified:

$$\frac{t \simeq_{\mathcal{L}} u}{U_t \equiv U_u}$$

where $t \simeq_{\mathcal{L}} u$ iff, for all valuation $\mu : \mathcal{V} \to \mathbb{N}$, $[\![t]\!]_\mu = [\![u]\!]_\mu$

| sym | $[\![\ ]\!]$ |
|---|---|
| $\mathtt{z}$ | 0 |
| $\mathtt{s}$ | $+1$ |
| $\sqcup$ | max |

# Problem: how to decide $\simeq_{\mathcal{L}}$ $\boxed{\text{in}}$ $\lambda\Pi/\mathcal{R}$ ?

$\simeq_{\mathcal{L}}$ is decidable (it is in Presburger arithmetic)

**but can we find:**

- a $\lambda\Pi$ signature $\Sigma$
- a *convergent* rewrite system $\mathcal{R}$ on $\Sigma$
- an encoding function $|\_| : \mathcal{L} \to \lambda\Pi/\mathcal{R}$

**such that:**

$$t \simeq_{\mathcal{L}} u \quad \text{iff} \quad |t| \hookrightarrow^*_{\mathcal{R}} {}^*_{\mathcal{R}}\hookleftarrow |u| \ ?$$

# Problem: how to decide $\simeq_{\mathcal{L}}$ $\boxed{\text{in}}$ $\lambda\Pi/\mathcal{R}$ ?

$\simeq_{\mathcal{L}}$ is decidable (it is in Presburger arithmetic)

**but can we find:**

- a $\lambda\Pi$ signature $\Sigma$
- a *convergent* rewrite system $\mathcal{R}$ on $\Sigma$
- an encoding function $|\_| : \mathcal{L} \to \lambda\Pi/\mathcal{R}$

**such that:**

$$t \simeq_{\mathcal{L}} u \quad \text{iff} \quad |t| \hookrightarrow^*_{\mathcal{R}} {}^*_{\mathcal{R}}\!\hookleftarrow |u| \ ?$$

**problem:**

$$\begin{array}{rcl}
(x \sqcup y) \sqcup z & \simeq_{\mathcal{L}} & x \sqcup (y \sqcup z) \\
x \sqcup y & \simeq_{\mathcal{L}} & y \sqcup x \\
x \sqcup \mathtt{s}^k x & \simeq_{\mathcal{L}} & \mathtt{s}^k x
\end{array}$$

where $\mathtt{s}^0 x = x$ and $\mathtt{s}^{k+1} x = \mathtt{s}(\mathtt{s}^k x)$

# Rewrite system on $\boxed{\text{closed}}$ levels/natural numbers $\mathtt{N}$

the previous equations are satisfied on closed terms by taking:

| $\Sigma$ | type | $[\![\ ]\!]$ |
|---|---|---|
| $0_{\mathtt{N}}$ | $\mathtt{N}$ | $0$ |
| $\mathtt{s}_{\mathtt{N}}$ | $\mathtt{N} \to \mathtt{N}$ | $+1$ |
| $\oplus$ | $\mathtt{N} \times \mathtt{N} \to \mathtt{N}$ | max |

$$\begin{array}{rcl} p \oplus 0_{\mathtt{N}} & \hookrightarrow & p \\ 0_{\mathtt{N}} \oplus q & \hookrightarrow & q \\ \mathtt{s}_{\mathtt{N}}\, p \oplus \mathtt{s}_{\mathtt{N}}\, q & \hookrightarrow & \mathtt{s}_{\mathtt{N}}\, (p \oplus q) \end{array}$$

# Rewrite system on $\boxed{\text{closed}}$ levels/natural numbers $\mathtt{N}$

the previous equations are satisfied on closed terms by taking:

| $\Sigma$ | type | $[\![\ ]\!]$ |
|---|---|---|
| $0_{\mathtt{N}}$ | $\mathtt{N}$ | $0$ |
| $\mathtt{s}_{\mathtt{N}}$ | $\mathtt{N} \to \mathtt{N}$ | $+1$ |
| $\oplus$ | $\mathtt{N} \times \mathtt{N} \to \mathtt{N}$ | max |

$$\begin{array}{rcl} p \oplus 0_{\mathtt{N}} & \hookrightarrow & p \\ 0_{\mathtt{N}} \oplus q & \hookrightarrow & q \\ \mathtt{s}_{\mathtt{N}}\, p \oplus \mathtt{s}_{\mathtt{N}}\, q & \hookrightarrow & \mathtt{s}_{\mathtt{N}}\, (p \oplus q) \end{array}$$

| $\Sigma$ | type | $[\![\ ]\!]$ |
|---|---|---|
| $+$ | $\mathtt{N} \times \mathtt{N} \to \mathtt{N}$ | $+$ |

$$\begin{array}{rcl} 0_{\mathtt{N}} + q & \hookrightarrow & q \\ \mathtt{s}_{\mathtt{N}}\, p + q & \hookrightarrow & \mathtt{s}_{\mathtt{N}}\, (p + q) \end{array}$$

# Canonical forms

every level $t$ with variables $x_1 < \ldots < x_n$ has:

- a unique **AC-canonical form** $t{\downarrow_{AC}}$

$$t \simeq_{AC} u \text{ iff } t{\downarrow_{AC}} = u{\downarrow_{AC}}$$

  assuming a total order on variables and terms (e.g. LPO)

  let $t \hookrightarrow^{!}_{AC} u$ iff $u = t{\downarrow_{AC}}$

# Canonical forms

every level $t$ with variables $x_1 < \ldots < x_n$ has:

- a unique **AC-canonical form** $t{\downarrow_{AC}}$

  $$t \simeq_{AC} u \text{ iff } t{\downarrow_{AC}} = u{\downarrow_{AC}}$$

  assuming a total order on variables and terms (e.g. LPO)

  let $t \hookrightarrow^{!}_{AC} u$ iff $u = t{\downarrow_{AC}}$

- a unique $\mathcal{L}$-**canonical form**

  $$\mathtt{s}^{k_0}\mathtt{z} \sqcup (\mathtt{s}^{k_1} x_1 \sqcup (\ldots \mathtt{s}^{k_n} x_n) \ldots) \text{ with } k_0 \geq k_1, \ldots, k_n$$

  where $\mathtt{s}^0 x = x$ and $\mathtt{s}^{k+1} x = \mathtt{s}(\mathtt{s}^k x)$

# Solution using rewriting with matching modulo AC

[Genestier, FSCD 2020]

$$t \simeq_{\mathcal{L}} u$$

iff

$$|t| \hookrightarrow^*_{\mathcal{R},AC} \simeq_{AC} \hookleftarrow^*_{\mathcal{R},AC} |u|$$

$$\begin{aligned}
|x| &= x \\
|\mathtt{z}| &= \mathtt{m}\,0\,\emptyset \\
|\mathtt{s}\,t| &= \mathtt{m}\,(\mathtt{s}\,0)\,(\mathtt{a}\,(\mathtt{s}\,0)\,|t|) \\
|u \sqcup v| &= \mathtt{m}\,0\,((\mathtt{a}\,0\,|u|) \cup (\mathtt{a}\,0\,|v|))
\end{aligned}$$

| $\Sigma$ | type | $[\![\ ]\!]$ |
|---|---|---|
| $\mathtt{m}$ | $\mathtt{N} \times \mathtt{E} \to \mathtt{L}$ | max |
| $\emptyset$ | $\mathtt{E}$ | $-\infty$ |
| $\mathtt{a}$ | $\mathtt{N} \times \mathtt{L} \to \mathtt{E}$ | $+$ |
| $\mathtt{A}$ | $\mathtt{N} \times \mathtt{E} \to \mathtt{E}$ | $+$ |
| $\cup$ | $\mathtt{E} \times \mathtt{E} \to \mathtt{E}$ | max |

with $\cup$ AC

$$\begin{aligned}
\mathtt{m}\,0\,(\mathtt{a}\,0\,x) &\hookrightarrow x \\
\mathtt{m}\,p\,(\mathtt{a}\,q\,(\mathtt{m}\,r\,X)) &\hookrightarrow \mathtt{m}\,(p \oplus (q+r))\,(\mathtt{A}\,q\,X) \\
\mathtt{m}\,p\,((\mathtt{a}\,q\,(\mathtt{m}\,r\,X)) \cup Y) &\hookrightarrow \mathtt{m}\,(p \oplus (q+r))\,((\mathtt{A}\,q\,X) \cup Y) \\
\mathtt{A}\,p\,\emptyset &\hookrightarrow \emptyset \\
\mathtt{A}\,p\,(\mathtt{a}\,q\,x) &\hookrightarrow \mathtt{a}\,(p+q)\,x \\
\mathtt{A}\,p\,(X \cup Y) &\hookrightarrow (\mathtt{A}\,p\,X) \cup (\mathtt{A}\,p\,Y) \\
X \cup \emptyset &\hookrightarrow X \\
(\mathtt{a}\,p\,x) \cup (\mathtt{a}\,q\,x) &\hookrightarrow \mathtt{a}\,(p \oplus q)\,x
\end{aligned}$$

example: if $p \geq q, r$ then $|\mathtt{s}^p\mathtt{z} \sqcup (\mathtt{s}^q x \sqcup \mathtt{s}^r y)| \hookrightarrow^* \mathtt{m}\,p\,(\mathtt{a}\,q\,x \cup \mathtt{a}\,r\,y)$

# Solution using rewriting with matching modulo AC

- matching modulo AC is NP-complete
- it doubles the size of the code [Férey, 2020]
- data structures for handling AC symbols efficiently do not combine very well with those for $\beta$-reduction and type checking

# Contribution: a solution not using matching modulo AC

Thm: $t \simeq_{\mathcal{L}} u$ iff $|t| \hookrightarrow^!_{AC} (\hookrightarrow_{\mathcal{R}} \hookrightarrow^!_{AC})^* (\hookleftarrow^!_{AC} \hookleftarrow_{\mathcal{R}})^* \hookleftarrow^!_{AC} |u|$

$$
\begin{aligned}
|x| &= \mathtt{S}\,0\,x \sqcup \mathtt{S}\,0\,\mathtt{z} \\
|\mathtt{z}| &= \mathtt{S}\,0\,\mathtt{z} \\
|\mathtt{s}\,t| &= \mathtt{S}\,(\mathtt{s}\,0)\,|t| \\
|u \sqcup v| &= |u| \sqcup |v|
\end{aligned}
$$

we replace every $\mathtt{s}$ by $\mathtt{S}(\mathtt{s}0)$ and every $x \in \mathcal{V} \cup \{\mathtt{z}\}$ by $\mathtt{S}0x \cup \mathtt{S}0\mathtt{z}$ ($\mathtt{S}kx$ is like $\mathtt{s}^k x$)

| $\Sigma$ | type | $[\![\ ]\!]$ |
|---|---|---|
| $\mathtt{z}$ | $\mathtt{L}$ | $0$ |
| $\mathtt{S}$ | $\mathtt{N} \times \mathtt{L} \to \mathtt{L}$ | $+$ |
| $\sqcup$ | $\mathtt{L} \times \mathtt{L} \to \mathtt{L}$ | max |

$$
\begin{aligned}
\mathtt{S}\,p\,(\mathtt{S}\,q\,x) &\hookrightarrow \mathtt{S}\,(p+q)\,x \\
\mathtt{S}\,p\,(x \sqcup y) &\hookrightarrow \mathtt{S}\,p\,x \sqcup \mathtt{S}\,p\,y \\
\mathtt{S}\,p\,x \sqcup \mathtt{S}\,q\,x &\hookrightarrow \mathtt{S}\,(p \oplus q)\,x \\
\mathtt{S}\,p\,x \sqcup (\mathtt{S}\,q\,x \sqcup y) &\hookrightarrow \mathtt{S}\,(p \oplus q)\,x \sqcup y
\end{aligned}
$$

**non-linear equations** are handled using any term ordering such that $\mathtt{S}px \leq \mathtt{S}qy$ iff $x < y$ or else $x = y$ and $p \leq q$ (LPO):

ex: $\mathtt{S}px \sqcup (\mathtt{S}ry \sqcup \mathtt{S}qx) \hookrightarrow^!_{AC} \mathtt{S}px \sqcup (\mathtt{S}qx \sqcup \mathtt{S}ry) \hookrightarrow_{\mathcal{R}} \mathtt{S}(p \oplus q)x \sqcup \mathtt{S}ry$

# Properties of $\hookrightarrow_{\mathcal{R}}\hookrightarrow^{!}_{AC}$

- $\hookrightarrow_{\mathcal{R}}\hookrightarrow^{!}_{AC}$ terminates

  Proof: $\hookrightarrow_{\mathcal{R}}\hookrightarrow^{!}_{AC}$ is included in $\hookrightarrow_{\mathcal{R}/AC}$ which can be proved terminating automatically by e.g. AProVE using polynomial interpretations checked by CeTA.

# Properties of $\hookrightarrow_{\mathcal{R}}\hookrightarrow^{!}_{AC}$

guarded = every occurrence of $x \in \mathcal{V} \cup \{\mathtt{z}\}$ is in a subterm $\mathtt{S}\, p\, x$
(ensured by the encoding and preserved by the rewrite rules)

- $\hookrightarrow_{\mathcal{R}}\hookrightarrow^{!}_{AC}$ is locally confluent
  on AC-canonical guarded terms with no variables of sort $\mathtt{N}$

  Proof. By hand. 10 critical pairs.
  Joinability requires associativity and commutativity of $+$ and $\oplus$
  and distributivity of $+$ over $\oplus$, which holds on closed terms of $\mathtt{N}$.

# Implementation of AC-canonization in Lambdapi using construction functions

https://github.com/Deducteam/lambdapi

following:

"On the implementation of construction functions for non-free concrete data types", with T. Hardin and P. Weis (ESOP 2007)

– AC-canonization is transparent since it is done at term construction time

– does not change the size of the code

# Implementation of AC-canonization in Lambdapi using construction functions

https://github.com/Deducteam/lambdapi

following:

"On the implementation of construction functions for non-free concrete data types", with T. Hardin and P. Weis (ESOP 2007)

- AC-canonization is transparent since it is done at term construction time

- does not change the size of the code

**Thank you! Questions?**