Encoding type universes without using matching modulo associativity and commutativity

Frédéric Blanqui

Deduction

EuroProofNet

4 August 2022
## Libraries of formal proofs today

<table>
<thead>
<tr>
<th>Library</th>
<th>Nb files</th>
<th>Nb objects*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coq Opam</td>
<td>16,000</td>
<td>473,000</td>
</tr>
<tr>
<td>Isabelle AFP</td>
<td>7,000</td>
<td>90,000</td>
</tr>
<tr>
<td>Lean Mathlib</td>
<td>2,000</td>
<td>81,000</td>
</tr>
<tr>
<td>Mizar Mathlib</td>
<td>1,400</td>
<td>77,000</td>
</tr>
<tr>
<td>HOL-Light</td>
<td>500</td>
<td>35,000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

* 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

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)
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.

\[
\begin{array}{cccc}
\text{system } A & \text{D}(A) & \text{D}(B) & \text{system } B \\
\text{ } & t & & u \\
\end{array}
\]
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. $B = array(x)$ for arrays of size $x$ |
| $\mathcal{R}$ | identification of types modulo a *convergent* rewrite system $\mathcal{R}$ |

<table>
<thead>
<tr>
<th>Terms</th>
<th>Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\star$</td>
<td>sort of types</td>
</tr>
<tr>
<td>$f$</td>
<td>global constant</td>
</tr>
<tr>
<td>$x$</td>
<td>local variable</td>
</tr>
<tr>
<td>$tu$</td>
<td>application</td>
</tr>
<tr>
<td>$\lambda x : t, u$</td>
<td>abstraction</td>
</tr>
<tr>
<td>$\Pi x : t, u$</td>
<td>dependent product</td>
</tr>
<tr>
<td>$t \rightarrow u$</td>
<td>abbrev. for $\Pi x : t, u$</td>
</tr>
<tr>
<td>when $x \notin u$</td>
<td></td>
</tr>
</tbody>
</table>

$\Gamma, x : A \vdash t : B \quad \Gamma \vdash \Pi x : A, B : \star$

$\Gamma \vdash \lambda x : A, t : \Pi x : A, B$

$\Gamma \vdash t : \Pi x : A, B \quad \Gamma \vdash u : A$

$\Gamma \vdash tu : B\{x \mapsto u\}$

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

$\Gamma \vdash t : B \quad \ldots$

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

$array(2 + 3) \equiv_{\beta\mathcal{R}} array(5)$
\[\lambda \Pi / R\] in practice: the Dedukti language

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

- Dedukti
- Isabelle
- HOL
- Matita
- Agda
- Coq
- FoCaLiZe
- Zenon
- ArchSAT
- TSTP
- Lambdapi
- PVS

Dedukti is a concrete language for \[\lambda \Pi / R\]
Lambdapi is a proof assistant for Dedukti

automated provers
Vampire, E, ...
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 \( \iota \rightarrow \omicron \)
- sets of sets of ur elements are of type \( (\iota \rightarrow \omicron) \rightarrow \omicron \)
- ...
Universes

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

  $$A \in U, B \in U \quad \Rightarrow \quad A \rightarrow B \in U$$

  Example: the set $U_0$ of the simple types $\iota, \iota \rightarrow o, \ldots$

- 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 $U_0$ ⇒ they need to inhabit a new universe $U_1$ containing $U_0$

- by iteration we get an infinite sequence of nested universes

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

$$A \in U_i \quad B \in U_j$$

$$A \rightarrow B \in U_{\max(i,j)}$$

available in some proof assistants like Coq, Agda, Lean
Universe polymorphism

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

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

and universes having equivalent levels are identified:

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

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

<table>
<thead>
<tr>
<th>sym</th>
<th>[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>z</td>
<td>0</td>
</tr>
<tr>
<td>s</td>
<td>+1</td>
</tr>
<tr>
<td>□</td>
<td>max</td>
</tr>
</tbody>
</table>
Problem: how to decide $\simeq_{\mathcal{L}} \text{ in } \lambda\Pi/\mathcal{R}$?

$\simeq_{\mathcal{L}}$ 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 $|\cdot| : \mathcal{L} \to \lambda\Pi/\mathcal{R}$

such that:

$$t \simeq_{\mathcal{L}} u \iff |t| \xrightarrow{\mathcal{R}}^* \xleftarrow{\mathcal{R}}^* |u| ?$$
Problem: how to decide $\approx_{\mathcal{L}} \text{ in } \lambda \Pi / \mathcal{R}$?

$\approx_{\mathcal{L}}$ 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:

$$t \approx_{\mathcal{L}} u \iff |t| \xrightarrow{\mathcal{R}}^* \xleftarrow{\mathcal{R}}^* |u| ?$$

problem:

$$(x \sqcup y) \sqcup z \approx_{\mathcal{L}} x \sqcup (y \sqcup z)$$
$$x \sqcup y \approx_{\mathcal{L}} y \sqcup x$$
$$x \sqcup s^k x \approx_{\mathcal{L}} s^k x$$

where $s^0 x = x$ and $s^{k+1} x = s(s^k x)$
Rewrite system on closed levels/natural numbers $\mathbb{N}$

The previous equations are satisfied on closed terms by taking:

$$
\begin{array}{|c|c|c|c|c|}
\hline
\Sigma & \text{type} & \llbracket \rrbracket \\
\hline
0_N & \mathbb{N} & 0 \\
S_N & \mathbb{N} \rightarrow \mathbb{N} & +1 \\
\oplus & \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} & \text{max} \\
\hline
\end{array}
$$

- $p \oplus 0_N \mapsto p$
- $0_N \oplus q \mapsto q$
- $S_N \ p \oplus S_N \ q \mapsto S_N \ (p \oplus q)$
Rewrite system on closed levels/natural numbers $\mathbb{N}$

the previous equations are satisfied on closed terms by taking:

$$
\begin{array}{|c|c|c|}
\hline
\Sigma & \text{type} & [ ] \\
\hline
0_{\mathbb{N}} & \mathbb{N} & 0 \\
\text{s}_{\mathbb{N}} & \mathbb{N} \to \mathbb{N} & +1 \\
\oplus & \mathbb{N} \times \mathbb{N} \to \mathbb{N} & \text{max} \\
\hline
\end{array}

\begin{align*}
0_{\mathbb{N}} \oplus 0_{\mathbb{N}} & \leftrightarrow p \\
0_{\mathbb{N}} \oplus q & \leftrightarrow q \\
\text{s}_{\mathbb{N}} p \oplus \text{s}_{\mathbb{N}} q & \leftrightarrow \text{s}_{\mathbb{N}} (p \oplus q) \\
0_{\mathbb{N}} + q & \leftrightarrow q \\
\text{s}_{\mathbb{N}} p + q & \leftrightarrow \text{s}_{\mathbb{N}} (p + q)
\end{align*}

$$
Canonical forms

every level $t$ with variables $x_1 < ... < x_n$ has:

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

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

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

let $t \leftrightarrow^!_{AC} u \text{ iff } u = t\downarrow_{AC}$
Canonical forms

every level $t$ with variables $x_1 < ... < x_n$ 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 $t \leftrightarrow_{AC}^! u \text{ iff } u = t \downarrow_{AC}$

- a unique **$L$-canonical form**

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

  where $s^0x = x$ and $s^{k+1}x = s(s^kx)$
Solution using rewriting with matching modulo AC

[Genestier, FSCD 2020]

\[
t \simeq_{L} u
\]

iff

\[
|t| \xrightarrow{\mathcal{R}, AC} AC \simeq AC \xleftarrow{\mathcal{R}, AC} |u|
\]

| \begin{align*}
|x| &= x \\
|z| &= m0\emptyset \\
|st| &= m(s0)(a(s0)|t|) \\
|u \sqcup v| &= m0((a0|u|) \cup (a0|v|))
\end{align*} |

\[
\begin{array}{|c|c|c|}
\hline
\Sigma & \text{type} & \text{max} \\
\hline
m & N \times E \to L & \text{max} \\
\emptyset & E & -\infty \\
a & N \times L \to E & + \\
A & N \times E \to E & + \\
\cup & E \times E \to E & \text{max} \\
\hline
\end{array}
\]

with \(\cup\) AC

\[
\begin{align*}
\text{m}0(a0x) & \mapsto x \\
m p(aq(mrX)) & \mapsto m(p \oplus (q + r))(AqX) \\
m p((aq(mrX)) \cup Y) & \mapsto m(p \oplus (q + r))((AqX) \cup Y) \\
Ap\emptyset & \mapsto \emptyset \\
Ap(aqx) & \mapsto a(p + q)x \\
Ap(X \cup Y) & \mapsto (ApX) \cup (ApY) \\
X \cup \emptyset & \mapsto X \\
(apx) \cup (aqx) & \mapsto a(p \oplus q)x
\end{align*}
\]

example: if \(p \geq q, r\) then

\[
|s^p z \cup (s^q x \cup s^r y)| \xrightarrow{\ast} m p(aq x \cup ar y)
\]
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: $t \simeq L u$ iff $|t| \hookrightarrow_{AC} (\hookrightarrow_{R} \hookrightarrow_{AC})^* (\leftarrow_{AC} \hookrightarrow_{R})^* \leftarrow_{AC} |u|

$|x| = S0x \sqcup S0z$

$|z| = S0z$

$|st| = S(s0)|t|$

$|u \sqcup v| = |u| \sqcup |v|$

we replace every $s$ by $S(s0)$ and every $x \in V \cup \{z\}$ by $S0x \sqcup S0z$

($Skx$ is like $s^k x$)

<table>
<thead>
<tr>
<th>$\Sigma$</th>
<th>type</th>
<th>$[\quad]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z$</td>
<td>L</td>
<td>0</td>
</tr>
<tr>
<td>$S$</td>
<td>$N \times L \rightarrow L$</td>
<td>$+$</td>
</tr>
<tr>
<td>$\sqcup$</td>
<td>$L \times L \rightarrow L$</td>
<td>max</td>
</tr>
</tbody>
</table>

$S\ p\ (S\ q\ x) \hookrightarrow S\ (p + q)\ x$

$S\ p\ (x \sqcup y) \hookrightarrow S\ p\ x \sqcup S\ p\ y$

$S\ p\ x \sqcup S\ q\ x \hookrightarrow S\ (p \oplus q)\ x$

$S\ p\ x \sqcup (S\ q\ x \sqcup y) \hookrightarrow S\ (p \oplus q)\ x \sqcup y$

**non-linear equations** are handled using any term ordering such that $Spx \leq Sqy$ iff $x < y$ or else $x = y$ and $p \leq q$ (LPO):

ex: $Spx \sqcup (Sry \sqcup Sqx) \leftrightarrow_{AC} Spx \sqcup (Sqx \sqcup Sry) \leftrightarrow_{R} S(p \oplus q)x \sqcup Sry$
Properties of $\rightarrow R \rightarrow^!_{AC}$

$\rightarrow R \rightarrow^!_{AC}$ terminates

Proof: $\rightarrow R \rightarrow^!_{AC}$ is included in $\rightarrow R/AC$ which can be proved terminating automatically by e.g. AProVE using polynomial interpretations checked by CeTA.
Properties of $\mathrel{\rightarrow^\diamond_{AC}}$

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

$\mathrel{\rightarrow^\diamond_{AC}}$ is locally confluent
on AC-canonical guarded terms with no variables of sort $\mathbb{N}$

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 $\mathbb{N}$. 
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?