Reengineering proofs in Dedukti: An example

Gilles Dowek

Inria and École normale supérieure de Paris-Saclay, 61, avenue du Président Wilson, 94235 Cachan Cedex, France, gilles.dowek@ens-cachan.fr.

The system Dedukti is a logical framework, implementing the $\lambda\Pi$-calculus modulo rewriting. It has been used to recheck proofs developed in several systems: iProver, Zenon, FoCaLiZe, HOL Light, and Matita [1]. To illustrate how it can be used to reengineer proofs, we consider the example of the translation to Simple type theory of proofs expressed in the Calculus of constructions.

1 Simple type theory

Simple type theory is expressed in Dedukti [1] with eight symbols and three rewrite rules:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>$\text{Type}$</td>
</tr>
<tr>
<td>$o$</td>
<td>$\text{type}$</td>
</tr>
<tr>
<td>$\iota$</td>
<td>$\text{type}$</td>
</tr>
<tr>
<td>arrow</td>
<td>$\text{type} \rightarrow \text{type} \rightarrow \text{type}$</td>
</tr>
<tr>
<td>$\eta$</td>
<td>$\text{type} \rightarrow \text{Type}$</td>
</tr>
<tr>
<td>$\Rightarrow$</td>
<td>$\eta(o) \rightarrow \eta(o) \rightarrow \eta(o)$</td>
</tr>
<tr>
<td>$\forall$</td>
<td>$\Pi x : \text{type} \ ((\eta(a) \rightarrow \eta(o)) \rightarrow \eta(o))$</td>
</tr>
<tr>
<td>$\varepsilon$</td>
<td>$\eta(o) \rightarrow \text{Type}$</td>
</tr>
</tbody>
</table>

Let us give examples showing how types, terms, and proofs of Simple type theory can be expressed in Dedukti. First, the simple types are expressed as Dedukti terms of type type. To do so, four symbols type, $o$, $\iota$, and arrow are declared. For instance the simple type $\iota \rightarrow \iota$ is expressed as the Dedukti term $\text{arrow}(\iota, \iota)$ of type type. This term is not a Dedukti type so, in Dedukti, no term can have the type $\text{arrow}(\iota, \iota)$. Thus, we declare a symbol $\eta$ mapping Dedukti terms of type type to Dedukti terms of type Type. So $\eta(\text{arrow}(\iota, \iota))$ is a Dedukti type. With the first rewrite rule, it reduces to $\eta(\iota) \rightarrow \eta(\iota)$.

The term $\lambda x : \iota \ x$ of Simple type theory, whose type, in Simple type theory, is $\iota \rightarrow \iota$, is then expressed as the Dedukti term $\lambda x : \eta(\iota) \ x$. In Dedukti, this term has type $\eta(\iota) \rightarrow \eta(\iota)$, that is $\eta(\text{arrow}(\iota, \iota))$. In the same way, the terms of Simple type theory of type $o$, the propositions, are expressed as Dedukti terms of type $\eta(o)$. To do so, two more symbols $\Rightarrow$ and $\forall$ are declared. For instance the proposition $\forall o \ \lambda x : o \ (x \Rightarrow x)$ is expressed in Dedukti as $(\forall o \ \lambda x : \eta(o) \ (\Rightarrow \ x \ x))$.

Finally, a symbol $\varepsilon$ is introduced that maps a proposition to the type of its proofs. So, the Dedukti type $\varepsilon(\forall o \ \lambda x : \eta(o) \ (\Rightarrow \ x \ x))$ is the type of the proofs of the proposition $\forall o \ \lambda x : o \ (x \Rightarrow x)$. With the two last rewrite rules, this type reduces to $\Pi x : \eta(o) \ (\varepsilon(x) \Rightarrow \varepsilon(x))$. The usual natural deduction proof of this proposition can be expressed as a lambda-term $\lambda x : \eta(o) \ \lambda y : \varepsilon(x) \ y$ that has type $\Pi x : \eta(o) \ (\varepsilon(x) \rightarrow \varepsilon(x))$, that is $\varepsilon(\forall o \ \lambda x : \eta(o) \ (\Rightarrow \ x \ x))$. 


2 The Calculus of constructions

2.1 A Pure type system

We consider the Calculus of constructions as a Pure type system \[2\] with two sorts \(\text{Prop}\) and \(\text{Type}\), an axiom \(\text{Prop} : \text{Type}\) and four rules \(\langle \text{Prop}, \text{Prop}, \text{Prop}\rangle, \langle \text{Prop}, \text{Type}, \text{Type}\rangle, \langle \text{Type}, \text{Prop}, \text{Prop}\rangle, \text{and} \langle \text{Type}, \text{Type}, \text{Type}\rangle\).

As we want the base types \(\iota, \text{nat}, \text{etc.}\) to be of type \(\text{Type}\), just like \(\iota \rightarrow \text{Prop}\), and not of type \(\text{Prop}\), we need to be able to declare variables of type \(\text{Type}\). To do so, we add a sort \(\text{Kind}\), an axiom \(\text{Type} : \text{Kind}\), but no rules involving \(\text{Kind}\) \[4\]. The term \(\text{Type}\) is the only term of type \(\text{Kind}\).

2.2 Translation

As any Pure type system, the Calculus of constructions can be expressed in \(\text{Dedukti}\) \[3\].

\[
\begin{align*}
U_{\text{Prop}} &: \text{Type} \\
U_{\text{Type}} &: \text{Type} \\
U_{\text{Kind}} &: \text{Type} \\
\varepsilon_{\text{Prop}} &: U_{\text{Prop}} \rightarrow \text{Type} \\
\varepsilon_{\text{Type}} &: U_{\text{Type}} \rightarrow \text{Type} \\
\varepsilon_{\text{Kind}} &: U_{\text{Kind}} \rightarrow \text{Type} \\
\text{Prop} &: U_{\text{Type}} \\
\text{Type} &: U_{\text{Kind}} \\
\Pi_{\langle \text{Prop}, \text{Prop}, \text{Prop}\rangle} &: \Pi X : U_{\text{Prop}} (((\varepsilon_{\text{Prop}} X) \rightarrow U_{\text{Prop}}) \rightarrow U_{\text{Prop}}) \\
\Pi_{\langle \text{Prop}, \text{Type}, \text{Type}\rangle} &: \Pi X : U_{\text{Prop}} (((\varepsilon_{\text{Prop}} X) \rightarrow U_{\text{Type}}) \rightarrow U_{\text{Type}}) \\
\Pi_{\langle \text{Type}, \text{Prop}, \text{Prop}\rangle} &: \Pi X : U_{\text{Type}} (((\varepsilon_{\text{Type}} X) \rightarrow U_{\text{Prop}}) \rightarrow U_{\text{Prop}}) \\
\Pi_{\langle \text{Type}, \text{Type}, \text{Type}\rangle} &: \Pi X : U_{\text{Type}} (((\varepsilon_{\text{Type}} X) \rightarrow U_{\text{Type}}) \rightarrow U_{\text{Type}}) \\
\varepsilon_{\text{Type}}(\text{Prop}) &: U_{\text{Prop}} \\
\varepsilon_{\text{Kind}}(\text{Type}) &: U_{\text{Type}} \\
\varepsilon_{\text{Prop}}(\Pi_{\langle \text{Prop}, \text{Prop}, \text{Prop}\rangle} X Y) &: \Pi x : (\varepsilon_{\text{Prop}} X) (\varepsilon_{\text{Prop}} (Y x)) \\
\varepsilon_{\text{Type}}(\Pi_{\langle \text{Prop}, \text{Type}, \text{Type}\rangle} X Y) &: \Pi x : (\varepsilon_{\text{Prop}} X) (\varepsilon_{\text{Type}} (Y x)) \\
\varepsilon_{\text{Prop}}(\Pi_{\langle \text{Type}, \text{Prop}, \text{Prop}\rangle} X Y) &: \Pi x : (\varepsilon_{\text{Type}} X) (\varepsilon_{\text{Prop}} (Y x)) \\
\varepsilon_{\text{Type}}(\Pi_{\langle \text{Type}, \text{Type}, \text{Type}\rangle} X Y) &: \Pi x : (\varepsilon_{\text{Type}} X) (\varepsilon_{\text{Type}} (Y x)) \\
\end{align*}
\]

This theory can be simplified in three ways: first we can write \(\text{type}\) for \(U_{\text{Type}}\), \(\circ\) for \(\text{Prop}\), \(\eta\) for \(\varepsilon_{\text{Type}}\), \(\varepsilon\) for \(\varepsilon_{\text{Prop}}\), \(\text{arrow}'\) for \(\Pi_{\langle \text{Type}, \text{Type}, \text{Type}\rangle}\), \(\Rightarrow'\) for \(\Pi_{\langle \text{Prop}, \text{Prop}, \text{Prop}\rangle}\), \(\forall\) for \(\Pi_{\langle \text{Type}, \text{Prop}, \text{Prop}\rangle}\), \(\pi\) for \(\Pi_{\langle \text{Prop}, \text{Type}, \text{Type}\rangle}\). Then, as \(\text{Type}\) only term of type \(\text{Kind}\), we can drop the symbols \(U_{\text{Kind}}, \varepsilon_{\text{Kind}}, \text{and} \text{Type}\) and the rule \(\varepsilon_{\text{Kind}}(\text{Type}) \rightarrow U_{\text{Type}}\). Finally, we
can replace $U_{prop}$ with $\eta(o)$ dropping rule: $\eta(o) \rightarrow U_{prop}$. We get this way

\[
\begin{align*}
type & : Type \\
o & : type \\
\eta & : type \rightarrow Type \\
\varepsilon & : \eta(o) \rightarrow Type \\
arrow' & : \Pi a : type (((\eta a) \rightarrow type) \rightarrow type) \\
\Rightarrow' & : \Pi p : \eta(o) (((\varepsilon p) \rightarrow \eta(o)) \rightarrow \eta(o)) \\
\forall & : \Pi a : type (((\eta a) \rightarrow \eta(o)) \rightarrow \eta(o)) \\
\pi & : \Pi p : \eta(o) (((\varepsilon p) \rightarrow type) \rightarrow type) \\
\end{align*}
\]

\[
\begin{align*}
\eta(\text{arrow'} a b) & \rightarrow \Pi x : (\eta a) (\eta b) \\
\varepsilon(\Rightarrow' p q) & \rightarrow \Pi x : (\varepsilon p) (\varepsilon q) \\
\varepsilon(\forall a p) & \rightarrow \Pi x : (\eta a) (\varepsilon(p) x) \\
\eta(\pi p a) & \rightarrow \Pi x : (\varepsilon p) (\eta a) \\
\end{align*}
\]

3 **Comparison**

The expression of the Calculus of constructions is similar to that of Simple type theory, with three differences. First, the symbol $\text{arrow}$ of type $type \rightarrow type \rightarrow type$ is replaced with a symbol $\text{arrow}'$ of type $\Pi a : type (((\eta a) \rightarrow type) \rightarrow type)$ and the rewrite rule

\[
\eta(\text{arrow'} a b) \rightarrow \Pi x : (\eta a) (\eta b)
\]

is transformed accordingly into

\[
\eta(\text{arrow'} a b) \rightarrow \Pi x : \eta(a) \eta(b)
\]

This difference reflects that, unlike Simple type theory, the Calculus of constructions has dependent functional types, like the type of a function that maps a natural number $n$ to an array of length $n$.

In the same way, the implication symbol $\Rightarrow$ of type $\eta(o) \rightarrow \eta(o) \rightarrow \eta(o)$ is replaced by a symbol $\Rightarrow'$ of type $\Pi p : \eta(o) (((\varepsilon(p) \rightarrow \eta(o)) \rightarrow \eta(o))$ and the rewrite rule

\[
\varepsilon(\Rightarrow p q) \rightarrow \Pi x : \varepsilon(p) \varepsilon(q)
\]

is transformed into

\[
\varepsilon(\Rightarrow' p q) \rightarrow \Pi x : \varepsilon(p) \varepsilon(q)
\]

This difference reflects that, unlike Simple type theory, the Calculus of constructions has dependent implications: in the implication $A \Rightarrow B$, the proposition $B$ may express a property of the proof of $A$.

Finally, the expression of the Calculus of constructions has an extra symbol $\pi$ of type $\Pi p : \eta(o) (((\varepsilon(p) \rightarrow type) \rightarrow type)$ and a reduction rule for this symbol

\[
\eta(\pi p a) \rightarrow \Pi x : \varepsilon(p) \eta(a)
\]

This difference reflects that, unlike in Simple type theory, in the Calculus of constructions, it is possible to express a function mapping a proof to a term.
4  Reengineering proofs

A proof in Simple type theory can always be translated to the Calculus of constructions, but a proof in the Calculus of constructions can only be translated to Simple type theory, if it does not use these three extra features of the Calculus of constructions.

In concrete examples, dependent types for functions are rarely used, dependent implication never, and functions mapping proofs to terms are used in very specific cases, that can for instance be replaced, in Simple type theory, with the use of the axiom of choice. Moreover, in the Calculus of constructions, we need a rule $\langle \text{Type, Kind, Kind} \rangle$ to be able to declare a variable $\text{array}$ of type $\text{nat} \rightarrow \text{Type}$ and then a type $\text{array } n$. In absence of such a rule, the symbol $\text{arrow}'$ is never dependent [4].

More precisely, we can define a theory that would contain not only the symbols $\text{arrow}$ and $\Rightarrow$, but also $\text{arrow}'$, $\Rightarrow'$ and $\pi$ and their associated rewrite rules. Each time, the dependent symbols $\text{arrow}'$ and $\Rightarrow'$ are used in a non dependent way, we can “improve” the proof, by replacing these symbols with $\text{arrow}$ and $\Rightarrow$, respectively. When such an improved proof does not contain the symbols $\text{arrow}'$, $\Rightarrow'$ and $\pi$ anymore, it happens to be expressed in Simple type theory.

References