A Rewrite System for Proof Constructivization

Raphaël Cauderlier
Inria, CNAM
raphael.cauderlier@inria.fr

Abstract

Proof constructivization is the problem of automatically extracting constructive proofs out of classical proofs. This process is required when classical theorem provers are integrated in intuitionistic proof assistants. We use the ability of rewrite systems to represent partial functions to implement heuristics for proof constructivization in Dedukti, a logical framework based on rewriting in which proofs are first-class objects which can be the subject of computation. We benchmark these heuristics on the proofs output by the automated theorem prover Zenon on the TPTP library of problems.

1. Introduction

Intuitionistic logic is usually presented as the fragment of classical logic obtained by removing the Law of Excluded Middle (or equivalent principles such as the Law of Double Negation) from the primitive axioms. Interestingly, it can also be seen as a supersystem of classical logic in the sense that classical formulae and proofs can be translated in intuitionistic logic thanks to double-negation translations (Kolmogorov 1925; Gödel 1933; Gentzen 1974; Kuroda 1951; Krivine 1990; Boudard and Hermant 2013; Dowek 2015; Gilbert 2015).

Unfortunately, neither point of view is very practical when we want to use a classical theorem prover together with a constructive proof assistant. In the first interpretation, the classical prover is only usable if the Law of Excluded Middle is added as an axiom in the proof assistant thus limiting the interpretation of the proof as an algorithm. In the second interpretation, the classical prover is seen as only able to produce proofs for formulae belonging to a fragment of the syntax where double-negations are mandatory at certain positions.

In practice, classical provers often use the refutation method which consists in adding the negation of the goal as hypothesis and trying to prove the inconsistency of the set of hypotheses. The justification for this simplification is exactly the Law of Double Negation, hence every proof coming from a refutation-based theorem prover contains at least one occurrence of a classical principle. However, a lot of automatically generated classical proofs are believed to be only accidentally classical in the sense that they use classical principles at non-critical places so constructive proofs can be extracted from them; we call this proof constructivization. One goal of this article is to give an experimental lower-bound on the number of proofs which can be constructivized.

Proof constructivization is an inherently incomplete activity. It obviously has to fail when the classically proved formula is not constructively provable but also when intuitionistic proofs of the formula require ingredients which are not present in the classical proof.

Type theory usually attach no computational behaviour to axioms. We propose however to interpret axioms such as the Law of Excluded Middle as partial functions defined by a set of rewrite rules; normalizing a proof relying on some axiom with respect to this rewrite system may (or not) lead to an axiom-free proof of the same theorem. Such rewrite systems can be defined in the $\lambda I$-calculus modulo (Saillard 2015), an extension of the Logical Framework $\lambda I$-calculus with rewriting. Concretely, we use Dedukti, an implementation of the $\lambda I$-calculus modulo which can be used to check proofs coming from the classical provers Zenon (Cauderlier and Halmagrand 2015) and iProver (Burel 2013).

This article starts with a short presentation of Dedukti in Section 2 then our notations for Natural Deduction proofs are given in Section 3 In Section 4 and 5 we define rewrite systems for proof constructivization. In Section 6 we show how they can be combined in Dedukti. An example of a successful constructivization by normalization is shown in Section 7 The article ends with experimental results on Zenon proofs for the TPTP library in Section 8 and a comparison with related works in Section 9.

2. Dedukti

Dedukti (Saillard 2015) is a type-checker for the $\lambda I$-calculus modulo, a logical framework based on rewriting. The typing rules for the $\lambda I$-calculus modulo are presented in Figure 1 It extends the $\lambda I$-calculus (Harper et al. 1993) (also known as LP and LF) by adding rewrite rules in the context and by replacing the congruence used in the conversion rule (which is $\beta$ equivalence in the $\lambda I$-calculus) by the congruence $\equiv_{\beta I}$ induced by $\beta$-reduction together with the rewrite rules which are present in the context. In order for type-checking to be decidable, a few assumptions are to be made:

- Left-hand sides of rewrite rules are restricted to linear higher-order patterns (Miller 1991). This constraint makes rewrite rule matching decidable. Dedukti checks this purely syntactic constraint.
- For each rewrite rule, a most general unifier must be computable by unification for the verification of the condition in the typing rule for rewrite rules. Thanks to this constraint, which is also checked by Dedukti, typing of rewrite rules is decidable.
- The congruence $\equiv_{\beta I}$ is approximated by syntactic comparison of $\beta I$ normal forms. So Dedukti might not terminate if $\beta I$-reduction is not terminating and it might be incomplete if $\beta I$-reduction is not confluent. Confluence is also a sufficient condition for subject reduction. These conditions are not decid-
Connective | Introduction Rule | Elimination Rule
--- | --- | ---
⊥ | ⊥ | ⊥
A ∧ B | (πA, πB) | fst(πA∧B) : A
| | snd(πA∧B) : B
A ∨ B | left(πA) | √\|\{πA⇒C, πB⇒C, πA∨B\} : C
| | right(πB)
∀x.P(x) | λx.πP(x) : A | πA⇒B(πA) : B
∃x.P(x) | (t, πP(t)) : A | πA⇒B(πA) : B

Figure 3. Intuitionistic Natural Deduction

Formulae are represented as types: the logical constants are interpreted as the singleton and the empty type, conjunction is cartesian product, disjunction is disjoint sum, implication is arrow, universal quantification is dependent product and existential quantification is dependent sum.

Proofs are interpreted as terms; we summarize our notations in Figure 3 where πA represents any term of type of A and HA represents a variable of type A.

4. Partial definitions of classical axioms

There are a lot of possible axiom schemes for turning intuitionistic logic into classical logic, we will focus on two of them:

• the Law of Excluded Middle: A ∨ ¬A
• the Law of Double Negation: ¬¬A ⇒ A

Contrary to other schemes such as Pierce’s law ((A ⇒ B) ⇒ A) ⇒ A, instantiating these schemes is done by providing just one formula. These schemes are equivalent but their instances are not: for a given formula A, A ∨ ¬A is constructively stronger than ¬¬A ⇒ A. Because of this, both schemes do not have the same computational behaviour.

4.1 A rewrite system for the Law of Excluded Middle

Let us abbreviate by LEM(A) the formula A ∨ ¬A. The following are constructive theorems proved in Figure 4:

• l0 : LEM(T)
• l1 : LEM(⊥)
• l2 : (LEM(A) ∧ LEM(B)) ⇒ LEM(A ∧ B)
• l3 : (LEM(A) ∧ LEM(B)) ⇒ LEM(A ∨ B)
• l4 : (LEM(A) ∧ LEM(B)) ⇒ LEM(A ⇒ B)

Thanks to these theorems, we can define a first rewrite system R1am pushing the classical axiom through the propositional connectives:

LEM(A) := A ∨ ¬A
lem(A) := LEM(A)
lem(T) ⇒ l0
lem(⊥) ⇒ l1
lem(A ∧ B) ⇒ l2(lem(A), lem(B))
lem(A ∨ B) ⇒ l3(lem(A), lem(B))
lem(A ⇒ B) ⇒ l4(lem(A), lem(B))

4.2 A rewrite system for the Law of Double Negation

We can do the same job for other classical axioms such as the Law of Double Negation. Let DN(A) abbreviate ¬¬A ⇒ A, the following are constructive theorems proved in Figure 5:

• d0 : DN(T)
Figure 4. Constructive instances of the Law of Excluded Middle

\[ l_0 : \text{LEM}(T) := \text{left}(T) \]
\[ l_1 : \text{LEM}(\bot) := \text{right}(\lambda H_\bot. \bot_\bot) \]
\[ l_2 : \text{LEM}(A \land B) := \text{right}(\lambda H_{A \land B}. H_\bot \text{fst}(H_{A \land B})) \]
\[ l_3 : \text{LEM}(A \land B) := \text{right}(\lambda H_{A \land B}. H_\bot \text{snd}(H_{A \land B})) \]
\[ H_{\text{LEM}(A)} := \text{right}(\lambda H_A. H_\bot) \]
\[ H_{\text{LEM}(B)} := \text{right}(\lambda H_B. H_\bot) \]
\[ H_{\text{LEM}(A \lor B)} := \lambda H_{A \lor B}. (H_A \lor H_B) \]
\[ H_{\text{LEM}(A \Rightarrow B)} := \lambda H_{A \Rightarrow B}. (H_A \Rightarrow H_B) \]
\[ H_{\text{LEM}(A /\Rightarrow B)} := \lambda H_{A /\Rightarrow B}. (H_A /\Rightarrow H_B) \]
\[ H_{\text{LEM}(A /\Rightarrow B)} := \lambda H_{A /\Rightarrow B}. (H_A /\Rightarrow H_B) \]

Figure 5. Constructive instances of the Law of Double Negation

- \[ d_0 : \text{DN}(T) := \lambda H_{\bot \bot}. \bot \]
- \[ d_1 : \text{DN}(\bot) := \lambda H_{\bot \bot}. \bot \]
- \[ d_2 : (\text{DN}(A) \land \text{DN}(B)) \Rightarrow \text{DN}(A \land B) \]
- \[ d_3 : \text{DN}(B) \Rightarrow \text{DN}(A \Rightarrow B) \]
- \[ d_4 : (\forall x. \text{DN}(P(x))) \Rightarrow \text{DN}(\forall x. P(x)) \]

This leads to the following rewrite system \( R_{\bot} \):

\[ \text{DN}(A) := \neg \neg A \Rightarrow A \]
\[ \text{dn}(A) := \text{DN}(A) \]
\[ \text{dn}(T) \leftarrow d_0 \]
\[ \text{dn}(\bot) \leftarrow d_1 \]
\[ \text{dn}(A \land B) \leftarrow d_2 (\text{dn}(A), \text{dn}(B)) \]
\[ \text{dn}(A \Rightarrow B) \leftarrow d_3 (\text{dn}(B)) \]
\[ \text{dn}(\forall x. P(x)) \leftarrow d_4 (\forall x. \text{dn}(P(x))) \]

These two rewrite systems are not very efficient at constructing proofs because they can do nothing smart on atoms. The rewrite system \( R_{\bot} \) is only able to constructivize proofs for formulae without atoms or quantifiers; it simply computes boolean values. The rewrite system \( R_{\bot} \) performs a bit better because the rewrite rule for implication \( A \Rightarrow B \) works for any \( A \); in particular \( \text{dn}(\neg A) \) reduces to a constructive proof so the rewrite system constructivizes proofs of double-negated formulae. Fortunately, we can go way further by inspecting the proof term.

5. Inspecting the proof

Seen as a function symbol in type theory, the symbol \( \text{dn} \) is a function of two parameters: the first one is a formula, the second one is a proof of the formula \( \neg \neg A \). The rewrite system that we have just presented only inspects the first argument \( A \) and acts independently of the second one. Thanks to higher-order rewriting, it is also possible to inspect the second one.

5.1 Two trivial special cases

Regardless of the shape of \( A \), there are two trivial ways in which a proof \( \pi_{\neg \neg A} \) of \( \neg \neg A \) can be constructivized into a proof of \( A \):

- seen as a function from \( \bot \) to \( \bot \), \( \pi_{\neg \neg A} \) does not use its argument, hence the current context is inconsistent so we can build a proof of \( A \)
- \( \pi_{\neg \neg A} \) is an instance of the constructive proof of \( \neg \neg A \) (which is \( \lambda H_{\bot \bot}. \bot \bot \equiv \bot \))

These two special cases can be written in Dedukti as higher-order rewrite rules:

\[ \text{dn}(A, \lambda H_{\bot \bot}. H_\bot) \rightarrow \text{id}_A (H_\bot) \quad (R_1) \]
\[ \text{dn}(A, \lambda H_{\bot \bot}. H_\bot) \rightarrow H_\bot \quad (R_2) \]

The second rule can be generalized by allowing any proof of \( \neg A \) depending on \( H_\bot \) (instead of exactly \( H_\bot \)):

\[ \text{dn}(A, \lambda H_{\bot \bot}. H_\bot) \rightarrow H_\bot \quad (R_2') \]

These rules restrict the positions in which \( H_\bot \) is allowed to appear; in order to favor their application, we consider proof transformations which make some proofs of \( \neg A \) disappear.

5.2 Eliminating negation proofs

The typical case where a proof of \( \neg A \) is useless is when it is eliminated to build a proof of \( A \):

\[ \bot \equiv \bot \quad (R_3) \]

In turn, to favor the application of this rewrite rule, we can give \( \bot \) some freedom by adding the usual rewrite rules which interpret elimination of \( \bot \) as error propagation:

\[ \bot \equiv \bot \quad (R_{\bot}) \]
\[ \lambda H_{\bot \bot}. \bot \equiv \bot \quad (R_{\bot}) \]
\[ \lambda H_{\bot \bot}. \bot \equiv \bot \quad (R_{\bot}) \]

Similar rewrite rules for all introduction and elimination rules can be added this way.

Another option for eliminating \( \bot \) is to make it progress toward the leaves in the hope that \( R_1 \) will be applicable in some branches and \( R_2 \) in others; this is the topic of next subsection.
5.3 Exchanging elimination rules

We further inspect the proof of \( \bot \) that missed to be captured by the pattern \( H_L \) in \( R_L \) and the pattern \( H_A \rightarrow H_A \) in \( R_L \) by looking at where it does use the hypothesis \( H_A \rightarrow \bot \) has no introduction rule so it can only be proved by an elimination rule. Elimination rules for disjunction and existential can be traversed by \( dn \) if the required proof of \( B \lor C \) and \( \exists x \, P(x) \) respectively do not use the assumption \( H_A \rightarrow \).

\[
\begin{align*}
\text{dn}(A, \lambda H \cdot \text{dn}_L) & \quad \text{v} \quad \text{r}(H_A \rightarrow B \rightarrow H_A, H_A \rightarrow \bot \rightarrow H_A, H_A \rightarrow C \rightarrow H_A, H_B \lor C) \quad \Rightarrow \\
\text{v} \quad \text{r}(H_B \lor C) & \quad \text{dn}(A, \lambda H \cdot \text{dn}_L) \quad \Rightarrow \\
\text{v} \quad \text{r}(H_B \lor C) & \quad \text{r}(A, \lambda H \cdot \text{dn}_L) \quad \Rightarrow \\
\text{v} \quad \text{r}(H_B \lor C) & \quad \text{r}(A, \lambda H \cdot \text{dn}_L) \quad \Rightarrow \\
\text{v} \quad \text{r}(H_B \lor C) & \quad \text{r}(A, \lambda H \cdot \text{dn}_L) \quad \Rightarrow \\
\text{v} \quad \text{r}(H_B \lor C) & \quad \text{r}(A, \lambda H \cdot \text{dn}_L)
\end{align*}
\]

\( dt \) is the rule controlling the strategy. It does however provide a command for printing a normal form of a term with respect to a rewrite system; this gives us two ways of combining two rewrite systems \( R_A \) and \( R_B \):

- **union**: we can ask Dedukti to compute \( \leftarrow_{R_A \cup R_B} \) by writing both systems in the same file
- **sequence**: by calling Dedukti twice, we can reduce terms using the relation \( \leftarrow_{R_1 \rightarrow R_2} \) that is we can ask for normal forms with respect to \( R_B \) of normal forms with respect to \( R_A \).

Moreover, the order in which the rewrite rules are given in a non-confluent Dedukti file is relevant: the earlier a rule is declared the higher its priority. For giving the rule \( R_3 \) priority over \( R_{\text{abort} \rightarrow \bot} \) and \( R_{\text{abort} \rightarrow \lambda} \), we just have to declare it first.

In which way to combine the rewrite systems of previous sections is a matter of heuristic choice; in practice, a good strategy consists in trying first the rules which remove axioms (\( R_1 \cup R_2 \)), then rules reducing the formula (\( R_3 \)), then rules pushing the axioms toward the leaves at the expance of exchanging the order of the elimination rules (the rewrite system of Subsection [5.3]) and finally the union of all the rewrite systems for \( dn \) presented in this paper together with cut-elimination rules.

6. Combining rewrite systems

As we have seen in Section [2], Dedukti is intended to be used with confluent rewrite systems so it does not provide a way for controlling the strategy. It does however provide a command for printing a normal form of a term with respect to a rewrite system; this gives us two ways of combining two rewrite systems \( R_A \) and \( R_B \):

- union: we can ask Dedukti to compute \( \leftarrow_{R_A \cup R_B} \) by writing both systems in the same file
- sequence: by calling Dedukti twice, we can reduce terms using the relation \( \leftarrow_{R_1 \rightarrow R_2} \) that is we can ask for normal forms with respect to \( R_B \) of normal forms with respect to \( R_A \).

Moreover, the order in which the rewrite rules are given in a non-confluent Dedukti file is relevant: the earlier a rule is declared the higher its priority. For giving the rule \( R_3 \) priority over \( R_{\text{abort} \rightarrow \bot} \) and \( R_{\text{abort} \rightarrow \lambda} \), we just have to declare it first.

In which way to combine the rewrite systems of previous sections is a matter of heuristic choice; in practice, a good strategy consists in trying first the rules which remove axioms (\( R_1 \cup R_2 \)), then rules reducing the formula (\( R_3 \)), then rules pushing the axioms toward the leaves at the expance of exchanging the order of the elimination rules (the rewrite system of Subsection [5.3]) and finally the union of all the rewrite systems for \( dn \) presented in this paper together with cut-elimination rules.

7. Example: Zenon classical proof of \( A \Rightarrow A \)

Once translated to Classical Natural Deduction, the classical proof of \( \neg A \Rightarrow A \) that comes out of Zenon is the following term:

\[
\text{dn}(A \Rightarrow A), \neg A \Rightarrow (A \Rightarrow A), \neg A \Rightarrow A \Rightarrow (A \Rightarrow A) \Rightarrow (A \Rightarrow A) \Rightarrow (A \Rightarrow A)
\]

Two rules apply here, the rule for implication from system \( R_A \) and the rule \( R_B \) for elimination of \( \bot \). Following the heuristic strategy of Section [6], the first step \( (R_1 \cup R_2) \) is skipped and we apply the rule in \( R_A \) leading to the following term:

\[
\text{dn}(A \Rightarrow A), \neg A \Rightarrow (A \Rightarrow A), \neg A \Rightarrow A \Rightarrow (A \Rightarrow A) \Rightarrow (A \Rightarrow A) \Rightarrow (A \Rightarrow A)
\]

we now need to unfold the definition of \( d_3 \):

\[
d_3(H_B(N_B), \neg A \Rightarrow (A \Rightarrow B)) := \lambda H_A \cdot H_B(N_B) (\lambda H_A \cdot \text{dn}(H_B(N_B), \neg A \Rightarrow (A \Rightarrow B)) \Rightarrow (A \Rightarrow B))
\]

so our proof of \( A \Rightarrow A \) is now

\[
\lambda H_A \cdot \text{dn}(A \Rightarrow A), \neg A \Rightarrow (A \Rightarrow A), \neg A \Rightarrow A \Rightarrow (A \Rightarrow A) \Rightarrow (A \Rightarrow A) \Rightarrow (A \Rightarrow A)
\]

we now perform elimination of \( \bot \) (\( R_3 \)), which has priority over \( \beta \)-reduction:

\[
\lambda H_A \cdot \text{dn}(A \Rightarrow A), \neg A \Rightarrow (A \Rightarrow A), \neg A \Rightarrow A \Rightarrow (A \Rightarrow A) \Rightarrow (A \Rightarrow A) \Rightarrow (A \Rightarrow A)
\]

\]

5.4 Confluence

The rules \( R_1 \) and \( R_2 \) are not confluent with the rewrite system \( R_{\text{dn}} \) of Section [2] for example, the term \( \lambda H \cdot \text{dn}_L(\tau, \lambda H \cdot \text{dn}_L) \) reduces to \( \lambda H \cdot \text{dn}_L(\tau, \lambda H \cdot \text{dn}_L) \) with respect to \( R_{\text{dn}} \) and to \( \lambda H \cdot \text{dn}_L(\tau, \lambda H \cdot \text{dn}_L) \) with respect to \( R_1 \). Even worse, the rules of Subsection [5.3] are to be used together but they form a non-confluent rewrite system: the term \( \bot \) reduces to both \( \bot \) (using \( R_3 \)) and \( \bot \) (using \( R_{\text{abort} \rightarrow \bot} \)).

In order to obtain the best behaviour out of our rewrite systems, we need to give them priorities.

\[
\lambda H_A \cdot \text{dn}(A \Rightarrow A), \neg A \Rightarrow (A \Rightarrow A), \neg A \Rightarrow A \Rightarrow (A \Rightarrow A) \Rightarrow (A \Rightarrow A) \Rightarrow (A \Rightarrow A)
\]
we now \( \beta \)-reduce:

\[
\lambda H_A. d_n(A, \lambda H_{\neg A}. H_{\neg A} A)
\]

and finally apply \( R_2 \), getting rid of the classical axiom:

\[
\lambda H_A. H_A
\]

As we can see, the translation to natural deduction has introduced a fortunate \( \beta \)-redex \((\lambda H_A', H_{\neg A}(A \Rightarrow A)) (\lambda H_A', H_A') H_A)\). Reducing this cut would forbid to fire \( R_3 \) but we need \( \beta \)-reduction to simplify the resulting proof so that \( R_2 \) can in turn be fired.

8. Experimental results

We have performed tests on the last version (v6.3.0) of the reference library for first-order problems: TPTP. This library contains 6528 problems for first-order logic (TPTP FOF format). We filtered these problems by running Zenon without asking for a proof term with a short timeout; Zenon claimed to have proved 1371 problems which form our starting benchmark. For every problems in this benchmark but two, Zenon provided a tableau proof that we type-checked in Dedukti. Among these 1369 tableau proofs, 1258 (91.9%) were translated to classical natural deduction. Among these natural deduction proofs, 1237 (98.3%) were normalized by the combination of rewrite systems presented in Section 6. All these normalized natural deduction proofs were rechecked in Dedukti and 778 (62.9% of the normalized classical proofs) were checked in intuitionistic natural deduction.

As a constructivization tool for natural deduction, our approach succeeded for 61.8% of the classical proofs.

The details for each TPTP category of problem are summarized in Section 8.

The experimental conditions for this study were the following:

- Processor: Intel Core i5-4310M @ 2.70GHz
- Timeout: 10 seconds for Zenon filtering phase, 10 minutes for each Dedukti call
- Tools versions: We used development versions of the tools built from their respective git repositories (git://scm.gforge.inria.fr/zenon/zenon.git for both Zenon and Zenonide). More precisely, Dedukti was built from branch develop (last commit: April 11th 2016), Zenon was built from branch modulo_intuit (last commit: February 5th 2016) and Zenonide was built from branch constructivisation (last commit: March 24th 2016).

<table>
<thead>
<tr>
<th>LK</th>
<th>NK</th>
<th>Normalized</th>
<th>NJ</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>17</td>
<td>17</td>
<td>13</td>
</tr>
<tr>
<td>23</td>
<td>13</td>
<td>11</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>91</td>
<td>91</td>
<td>87</td>
<td>57</td>
</tr>
<tr>
<td>213</td>
<td>210</td>
<td>210</td>
<td>184</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>62</td>
<td>62</td>
<td>62</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>28</td>
<td>8</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>39</td>
<td>38</td>
<td>35</td>
<td>22</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>101</td>
<td>92</td>
<td>92</td>
<td>78</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>24</td>
<td>24</td>
<td>24</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>135</td>
<td>135</td>
<td>135</td>
<td>102</td>
</tr>
<tr>
<td>84</td>
<td>80</td>
<td>78</td>
<td>60</td>
</tr>
<tr>
<td>21</td>
<td>21</td>
<td>21</td>
<td>20</td>
</tr>
<tr>
<td>43</td>
<td>43</td>
<td>43</td>
<td>1</td>
</tr>
<tr>
<td>132</td>
<td>132</td>
<td>131</td>
<td>75</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>264</td>
<td>201</td>
<td>194</td>
<td>62</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Total: 1369, 1258, 778

Each line is a TPTP category of problems:

- the \( \text{LK} \) column contains the number of problems proved by Zenon in classical sequent calculus
- the \( \text{NK} \) column contains the number of proofs translated in classical natural deduction
- the \( \text{Normalized} \) column contains the number of classical proofs which have been normalized with respect to our rewrite systems
- the \( \text{NJ} \) column contains the number of proofs for which constructivization has succeeded in a proof in intuitionistic natural deduction

Figure 6. Per-category Comparative results with Zenonide

9. Related Works

The differences between classical and intuitionistic logic have been deeply studied since the beginnings of intuitionistic logic leading to the discovery of double-negation translations and extensions of the Curry-Howard correspondence to classical logic. Concretely, a few automated theorem provers can be used for intuitionistic logic but their integration in intuitionistic proof assistants is far from easy because they do not yet provide proof certificates in a checkable format. We know only one exception to this rule: a constructivization module for Zenon called Zenonide which is able to produce proofs in Dedukti format.

9.1 Double-negation translations

It is usually easy to test whether a formula is in the image of a given double-negation translation. Since for such formulae intuitionistic provability corresponds to classical provability, this provides a simple criterion for proof constructivization which does not depend on the classical proof but only on the proven formula. This criterion is not very powerful but it is very efficient: typically in linear time and finite memory.

Our first rewrite system for the Law of Double Negation \( R_{dn} \) is related to the way one can replace a double-negation translation by another one. Because the right-hand side of the rule for \( dn(A \Rightarrow B) \) uses \( dn(B) \) but not \( dn(A) \), the correct way to look at \( R_{dn} \) is as a transformer for polarized double-negation translations (Boudard and Hermant, 2013).

In the particular case of Zenon tableaux proofs and their translation to Classical Natural Deduction, the \( dn \) classical axiom is used at the head of the proof and after introduction of universal quantification only. Because Zenon finds cut-free proofs in Sequent Calculus, the subformula property guarantees that all double negations corresponding to these classical axioms appear in positions where the polarized version of Gödel-Gentzen double-negation translation would also have added a double-negation. After normaliza-
tion by our rewrite system $R_{\text{dn}}$, they are placed at positions where a lighter translation, Gilbert’s double-negation translation (Gilbert 2015), would also put double-negations.

9.2 Intuitionistic provers

A few automated theorem provers for intuitionistic logic have been developed. The ILTP library (Kahs et al. 2005) is a benchmark constructed from the TPTP problems in FOT format (non-clausal first-order formulae) to evaluate intuitionistic provers. The most performant intuitionistic prover for predicate logic on this benchmark is by far the iLeanCoP prover. It is noticeable that iLeanCoP is built by far the iLeanCoP prover. It is noticeable that iLeanCoP is built.

The main difference between our work and an intuitionistic prover such as iLeanCoP is that, in case of failure, iLeanCoP can ask LeanCoP to provide another classical proof. For example, our technique fails to constructivize the following proof of $A \implies (A \lor \neg A)$: $\lambda H. \text{lem}(A)$. Backtracking makes however iLeanCoP complete for first-order intuitionistic logic. According to Otten (2005), this backtracking feature is rarely used because the first classical proof is usually constructive.

Unfortunately, intuitionistic provers such as iLeanCoP do not produce certificates so we can not easily integrate them in intuitionistic proof assistants.

9.3 Zenonide

Zenonide is a constructivization module for Zenon developed by Frédéric Gilbert. Because Zenonide has access to the internal representation of proofs in Zenon, it has access to the proof context for each proof node so the constructivization of Zenon proof of $A \implies A$ is trivial for Zenonide whereas we have seen in Section 7 that it required some work in our case.

Zenonide is however not able to backtrack to another classical proof as iLeanCoP so it is an interesting middle point between our approach to constructivization and intuitionistic proof search.

Zenonide does not need to translate the classical proof to Natural Deduction, it tries to transform a proof in Classical Sequent Calculus to a proof in Intuitionistic Sequent Calculus so it avoids the combinatorial explosion appearing with some problems of the syntactic category of TPTP which have been especially designed to have no small proof in Natural Deduction. The price to pay for using Sequent Calculus is that more commutations of deduction rules have to be taken into account.

Zenon, as many theorem provers, searches for cut-free proofs; this is a very good point for Zenonide since the cut rule in Sequent Calculus behaves very badly with proof constructivization. Our input proofs do however use natural deduction cuts and these cuts are, as we have seen, sometimes welcomed.

For all these reasons, Zenonide globally performs better than our rewrite system but also fails on some proofs that we manage to constructivize: on the set of 1371 problems proved by Zenon, Zenonide proves 915 problems constructively but a very fair amount of proofs (113) was constructivized by our rewrite systems despite Zenonide lacks to prove it. If we use our rewrite systems together with Zenonide, we obtain a total constructivization rate of 81.7%.

9.4 Extensions of the Curry-Howard correspondence for Classical Logic

The Curry-Howard correspondence, which is at the heart of the use of logical frameworks such as Dedukti for checking proofs, has been extended to classical reasoning in several ways.

Minimal logic can be extended to a classical logic of implication by Pierce Law ($((A \implies B) \implies A) \implies A$) which is a possible type for the call-cc control operator found in the Scheme programming language. This remark lead to Parigot’s $\lambda \mu$-calculus which corresponds to a classical extension of minimal natural-deduction where several formulae are allowed at the right side of sequents (Parigot 1992).

Classical Sequent Calculus has also been the subject of interpretations through the Curry-Howard correspondence leading to Curien and Herbelin’s $\lambda \mu \nu$-calculus for minimal classical sequent calculus (Curien and Herbelin 2000).

An interpretation of classical logic in terms of stack manipulations is also investigated in the context of classical realizability (Krivine 2001).

All these systems suffer, as we do, from a lack of confluence but this is directly connected with non-confluence of classical cut elimination whereas we do not even need to consider cut-elimination to loose confluence.

As extensions of typed $\lambda$-calculus, it is possible to ask in these systems whether a given term (that is, a classical proof) reduces to a pure $\lambda$-term (that is, a constructive proof). Conversely, the rewrite systems that we have proposed can be seen as alternative semantics or program transformations in these systems.

We believe that Dedukti is a good framework for studying extensions of these systems to predicate logic.

10. Conclusion

By combining relatively naive rewrite rules, we constructivized automatically a good proportion of the classical proofs that were found by the theorem prover Zenon.

Our approach can easily be generalized to extensions of Predicate Logic such as polymorphism and Deduction Modulo which are implemented in Zenon. These two extensions would allow the constructivization of program verification proofs used in FoCal, iZe, Why3, and Atelier B. We expect these proofs to be particularly easy to constructivize because the user of these systems is encouraged to define decidable predicates as functions to type bool and $\forall b : \text{bool}. \neg b \lor b$ is constructively provable so even the very simple rewrite system $R_{\text{dn}}$ together with a rewrite rule for the atomic case (for atoms of type bool) should successfully constructivize a lot of problems.

We can also improve our handling of failure by trying to reach an intermediate logic, an extension of intuitionistic logic with an axiom scheme which is classically provable, not intuitionistically provable but not strong enough to prove the Law of Excluded Middle either. Such an intermediate axiom scheme is $\neg A \lor \neg \neg A$ which allows some De Morgan simplification rules. Other axioms of interest would be decidability of equality and decidability of atoms.

We use Dedukti in a very different context than its common usage as a universal proof checker. We hope this new use, especially the usage of non-confluent intermediate rewrite systems producing proofs to be checked in a safe rewrite system (in this case, the signature for intuitionistic natural deduction, which contains no rewrite rule), will be a fruitful source for further automatic elimination of axioms. This topic is not only important for benefiting from the witness property of constructive proofs but also from a reverse mathematics point of view and for interoperating proofs in theories whose union is inconsistent.

References

