A Rewrite System for Proof Constructivization

Raphaël Cauderlier

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 attaches 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 $\lambda\Pi$-calculus modulo (Saillard 2015), an extension of the Logical Framework $\lambda\Pi$-calculus with rewriting. Concretely, we use Dedukti, an implementation of the $\lambda\Pi$-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 $\lambda\Pi$-calculus modulo, a logical framework based on rewriting closely related to Martin-Löf's Logical Framework (Nordstrom et al. 1989). The typing rules for the $\lambda\Pi$-calculus modulo are presented in Figure 1. It extends the $\lambda\Pi$-calculus (Harper et al. 1993) (also known as λP and LF) by adding rewrite rules in the context and by replacing the congruence used in the conversion rule (which is β equivalence in the $\lambda\Pi$-calculus) by the congruence $\equiv_{\beta\Gamma}$ induced by β-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 $\equiv_{\beta\Gamma}$ is approximated by syntactic comparison of $\beta\Gamma$ normal forms. So Dedukti might not terminate if $\beta\Gamma$-reduction is not terminating and it might be incomplete if $\beta\Gamma$-reduction is not confluent. Confluence is also a sufficient condition for subject reduction. These conditions are not decid-

Syntax

s	:=	**Type** | **Kind**	*sorts*
t	:=	$s \mid x \mid t\ t \mid \lambda x : t.\ t \mid \Pi x : t.\ t$	*terms*
Γ	:=	$\emptyset \mid \Gamma, x : t \mid \Gamma, t \hookrightarrow t$	*contexts*
Δ	:=	$\emptyset \mid \Delta, x : t$	*local contexts*

Typing

$$\frac{}{\emptyset \vdash}\ \text{(Empty)} \qquad \frac{\Gamma \vdash x : A \qquad x \notin \Gamma}{\Gamma, x : A \vdash}\ \text{(Decl)}$$

$$\frac{\forall(\sigma, \Delta, A).\ (\Gamma, \Delta \vdash l\sigma : A) \Rightarrow (\Gamma, \Delta \vdash r\sigma : A)}{\Gamma, l \hookrightarrow r \vdash}\ \text{(Rule)}$$

$$\frac{\Gamma \vdash}{\Gamma \vdash \textbf{Type} : \textbf{Kind}}\ \text{(Type)} \qquad \frac{\Gamma \vdash \qquad (x : A) \in \Gamma}{\Gamma \vdash x : A}\ \text{(Var)}$$

$$\frac{\Gamma \vdash f : \Pi x : A.\ B \qquad \Gamma \vdash a : A}{\Gamma \vdash f\ a : B\{x \backslash a\}}\ \text{(App)}$$

$$\frac{\Gamma \vdash A : \textbf{Type} \qquad \Gamma, x : A \vdash b : B}{\Gamma \vdash \lambda x : A.\ b : \Pi x : A.\ B}\ \text{(Abs)}$$

$$\frac{\Gamma \vdash A : \textbf{Type} \qquad \Gamma, x : A \vdash B : s}{\Gamma \vdash \Pi x : A.\ B : s}\ \text{(Prod)}$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash B : s \qquad A \equiv_{\beta\Gamma} B}{\Gamma \vdash t : B}\ \text{(Conv)}$$

Figure 1. $\lambda\Pi$-calculus modulo

$A, B, P, \ldots$	::=	$\mathcal{P}$	*atoms*
	|	$\top \mid \bot$	*constants*
	|	$A \wedge B \mid A \vee B \mid A \Rightarrow B$	*connectives*
	|	$\forall x.P(x) \mid \exists x.P(x)$	*quantifiers*

Figure 2. Syntax of formulae in Predicate logic

able but Dedukti can delegate confluence checking to dedicated tools.

3. Natural Deduction in Dedukti

Natural Deduction is a standard formalism for intuitionistic and classical proofs in Predicate logic. It is very close to type systems for the λ-calculus so it is a convenient proof system for a logical framework based on type theory such as Dedukti.

3.1 Syntax

We use the standard notations for formulae of Predicate logic (see Figure 2). Our basic connectives are constants, conjunction, disjunction, implication, and quantifiers. Negation and equivalence are seen as derived connectives defined by $\neg A := A \Rightarrow \bot$ and $A \Leftrightarrow B := (A \Rightarrow B) \wedge (B \Rightarrow A)$ respectively.

3.2 Intuitionistic Natural Deduction in Dedukti

In Dedukti, Intuitionistic Natural Deduction is easy to define following the Curry-Howard correspondence and the proposition-as-type discipline as it is usually done in Martin-Löf's Logical Framework (Nordstrom et al. 1989) for example.

Connective	Introduction Rule	Elimination Rule
$\top$	$\top_I$	
$\bot$		$\bot_E^A(\pi_\bot) : A$
$A \wedge B$	(π_A, π_B)	$\mathsf{fst}(\pi_{A\wedge B}) : A$ $\mathsf{snd}(\pi_{A\wedge B}) : B$
$A \vee B$	$\mathsf{left}(\pi_A)$ $\mathsf{right}(\pi_B)$	$\vee_E^C(\pi_{A\Rightarrow C}, \pi_{B\Rightarrow C}, \pi_{A\vee B}) : C$
$A \Rightarrow B$	$\lambda H_A.\pi_B$	$\pi_{A\Rightarrow B}(\pi_A) : B$
$\forall x.P(x)$	$\lambda x.\pi_{P(x)}$	$\pi_{\forall x.P(x)}(t) : P(t)$
$\exists x.P(x)$	$(t, \pi_{P(t)})$	$\exists_E^A(\pi_{\forall x.P(x)\Rightarrow A}, \pi_{\exists x.P(x)}) : A$

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 π_A represents any term of type of A and H_A 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: $A \vee \neg A$
- the Law of Double Negation: $\neg\neg A \Rightarrow A$

Contrary to other schemes such as Pierce's law $((A \Rightarrow B) \Rightarrow A) \Rightarrow A$, instantiating these schemes is done by providing just one formula. These schemes are equivalent but their instances are not: for a given formula A, $A \vee \neg A$ is constructively stronger than $\neg\neg A \Rightarrow A$. 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 $\mathsf{LEM}(A)$ the formula $A \vee \neg A$. The following are easy constructive theorems, their proofs are not very interesting but we give them in Figure 4 for the sake of completeness:

- $l_0 : \mathsf{LEM}(\top)$
- $l_1 : \mathsf{LEM}(\bot)$
- $l_2 : (\mathsf{LEM}(A) \wedge \mathsf{LEM}(B)) \Rightarrow \mathsf{LEM}(A \wedge B)$
- $l_3 : (\mathsf{LEM}(A) \wedge \mathsf{LEM}(B)) \Rightarrow \mathsf{LEM}(A \vee B)$
- $l_4 : (\mathsf{LEM}(A) \wedge \mathsf{LEM}(B)) \Rightarrow \mathsf{LEM}(A \Rightarrow B)$

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

$$\begin{array}{l}
\mathsf{LEM}(A) := A \vee \neg A \\
\mathsf{lem}(A) : \mathsf{LEM}(A) \\
\mathsf{lem}(\top) \hookrightarrow l_0 \\
\mathsf{lem}(\bot) \hookrightarrow l_1 \\
\mathsf{lem}(A \wedge B) \hookrightarrow l_2(\mathsf{lem}(A), \mathsf{lem}(B)) \\
\mathsf{lem}(A \vee B) \hookrightarrow l_3(\mathsf{lem}(A), \mathsf{lem}(B)) \\
\mathsf{lem}(A \Rightarrow B) \hookrightarrow l_4(\mathsf{lem}(A), \mathsf{lem}(B))
\end{array}$$

$$
\begin{array}{l}
l_0 : \mathsf{LEM}(\top) := \mathsf{left}(\top_I) \\
l_1 : \mathsf{LEM}(\bot) := \mathsf{right}(\lambda H_\bot.\ H_\bot) \\
l'_2(H_{\neg A}) : \mathsf{LEM}(A \wedge B) := \mathsf{right}(\lambda H_{A\wedge B}.\ H_{\neg A}\ \mathsf{fst}(H_{A\wedge B})) \\
l''_2(H_{\neg B}) : \mathsf{LEM}(A \wedge B) := \mathsf{right}(\lambda H_{A\wedge B}.\ H_{\neg B}\ \mathsf{snd}(H_{A\wedge B})) \\
l_2(H_{\mathsf{LEM}(A)}, H_{\mathsf{LEM}(B)}) : \mathsf{LEM}(A \wedge B) := \\
\quad \vee_E^{\mathsf{LEM}(A\wedge B)}(\lambda H_A.\ \vee_E^{\mathsf{LEM}(A\wedge B)}(\lambda H_B.\ \mathsf{left}(H_A, H_B), \\
\qquad\qquad\qquad\qquad\qquad\qquad \lambda H_{\neg B}.\ l''_2(H_{\neg B}), \\
\qquad\qquad\qquad\qquad\qquad\qquad H_{\mathsf{LEM}(B)}), \\
\qquad\qquad \lambda H_{\neg A}.\ l'_2(H_{\neg A}), \\
\qquad\qquad H_{\mathsf{LEM}(A)}) \\
l'_3(H_{\neg A}, H_{\neg B}) : \mathsf{LEM}(A \vee B) := \\
\quad \mathsf{right}(\lambda H_{A\vee B}.\ \vee_E^{\perp}(H_{\neg A}, H_{\neg B}, H_{A\vee B})) \\
l_3(H_{\mathsf{LEM}(A)}, H_{\mathsf{LEM}(B)}) : \mathsf{LEM}(A \vee B) := \\
\quad \vee_E^{\mathsf{LEM}(A\vee B)}(\lambda H_A.\ \mathsf{left}(\mathsf{left}(H_A)), \\
\qquad\qquad \lambda H_{\neg A}.\ \vee_E^{\mathsf{LEM}(A\vee B)}(\lambda H_B.\ \mathsf{left}(\mathsf{right}(H_B)), \\
\qquad\qquad\qquad\qquad\qquad\qquad \lambda H_{\neg B}.\ l'_3(H_{\neg A}, H_{\neg B}), \\
\qquad\qquad\qquad\qquad\qquad\qquad H_{\mathsf{LEM}(B)}), \\
\qquad\qquad H_{\mathsf{LEM}(A)}) \\
l'_4(H_B) : \mathsf{LEM}(A \Rightarrow B) := \mathsf{left}(\lambda H_A.\ H_B) \\
l''_4(H_A, H_{\neg B}) : \mathsf{LEM}(A \Rightarrow B) := \\
\quad \mathsf{right}(\lambda H_{A\Rightarrow B}.\ H_{\neg B}\ (H_{A\Rightarrow B}\ H_A)) \\
l'''_4(H_{\neg A}) : \mathsf{LEM}(A \Rightarrow B) := \mathsf{left}(\lambda H_A.\ \bot_E^B(H_{\neg A}\ H_A)) \\
l_4(H_{\mathsf{LEM}(A)}, H_{\mathsf{LEM}(B)}) : \mathsf{LEM}(A \Rightarrow B) := \\
\quad \vee_E^{\mathsf{LEM}(A\Rightarrow B)}(\lambda H_A.\ \vee_E^{\mathsf{LEM}(A\Rightarrow B)}(\lambda H_B.\ l'_4(H_B), \\
\qquad\qquad\qquad\qquad\qquad\qquad \lambda H_{\neg B}.\ l''_4(H_A, H_{\neg B}), \\
\qquad\qquad\qquad\qquad\qquad\qquad H_{\mathsf{LEM}(B)}), \\
\qquad\qquad \lambda H_{\neg A}.\ l'''_4(H_{\neg A}), \\
\qquad\qquad H_{\mathsf{LEM}(A)})
\end{array}
$$

Figure 4. Constructive instances of the Law of Excluded Middle

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 $\mathsf{LDN}(A)$ abbreviate $\neg\neg A \Rightarrow A$, the following are constructive theorems proved in Figure 5:

- $d_0 : \mathsf{LDN}(\top)$
- $d_1 : \mathsf{LDN}(\bot)$
- $d_2 : (\mathsf{LDN}(A) \wedge \mathsf{LDN}(B)) \Rightarrow \mathsf{LDN}(A \wedge B)$
- $d_3 : \mathsf{LDN}(B) \Rightarrow \mathsf{LDN}(A \Rightarrow B)$
- $d_4 : (\forall x.\ \mathsf{LDN}(P(x))) \Rightarrow \mathsf{LDN}(\forall x.\ P(x))$

This leads to the following rewrite system R_{ldn}:

$$
\begin{array}{l}
\mathsf{LDN}(A) := \neg\neg A \Rightarrow A \\
\mathsf{ldn}(A) : \mathsf{LDN}(A) \\
\mathsf{ldn}(\top) \hookrightarrow d_0 \\
\mathsf{ldn}(\bot) \hookrightarrow d_1 \\
\mathsf{ldn}(A \wedge B) \hookrightarrow d_2(\mathsf{ldn}(A), \mathsf{ldn}(B)) \\
\mathsf{ldn}(A \Rightarrow B) \hookrightarrow d_3(\mathsf{ldn}(B)) \\
\mathsf{ldn}(\forall x.\ P(x)) \hookrightarrow d_4(\lambda x.\ \mathsf{ldn}(P(x)))
\end{array}
$$

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

$$
\begin{array}{l}
d_0 : \mathsf{LDN}(\top) := \lambda H_{\neg\neg\top}.\ \top_I \\
d_1 : \mathsf{LDN}(\bot) := \lambda H_{\neg\neg\bot}.\ H_{\neg\neg\bot}\ (\lambda H_\bot.\ H_\bot) \\
d'_2(H_{\mathsf{LDN}(A)}, H_{\neg\neg(A\wedge B)}) : A := \\
\quad H_{\mathsf{LDN}(A)}\ (\lambda H_{\neg A}.\ H_{\neg\neg(A\wedge B)}\ (\lambda H_{A\wedge B}.\ H_{\neg A}\ \mathsf{fst}(H_{A\wedge B}))) \\
d''_2(H_{\mathsf{LDN}(B)}, H_{\neg\neg(A\wedge B)}) : B := \\
\quad H_{\mathsf{LDN}(B)}\ (\lambda H_{\neg B}.\ H_{\neg\neg(A\wedge B)}\ (\lambda H_{A\wedge B}.\ H_{\neg B}\ \mathsf{snd}(H_{A\wedge B}))) \\
d_2(H_{\mathsf{LDN}(A)}, H_{\mathsf{LDN}(B)}) : \mathsf{LDN}(A \wedge B) := \\
\quad \lambda H_{\neg\neg(A\wedge B)}. \\
\qquad (d'_2(H_{\mathsf{LDN}(A)}, H_{\neg\neg(A\wedge B)}), d''_2(H_{\mathsf{LDN}(B)}, H_{\neg\neg(A\wedge B)})) \\
d_3(H_{\mathsf{LDN}(B)}) : \mathsf{LDN}(A \Rightarrow B) := \\
\quad \lambda H_{\neg\neg(A\Rightarrow B)}.\lambda H_A.\ H_{\mathsf{LDN}(B)}\ (\lambda H_{\neg B}. \\
\qquad H_{\neg\neg(A\Rightarrow B)}\ (\lambda H_{A\Rightarrow B}.\ H_{\neg B}\ (H_{A\Rightarrow B}\ H_A))) \\
d_4(H_{\forall x.\ \mathsf{LDN}(P(x))}) : \mathsf{LDN}(\forall x.\ P(x)) := \\
\quad \lambda H_{\neg\neg(\forall x.\ P(x))}.\lambda x.\ H_{\forall x.\ \mathsf{LDN}(P(x))}\ x\ (\lambda H_{\neg P(x)}. \\
\qquad H_{\neg\neg(\forall x.\ P(x))}\ (\lambda H_{\forall x.\ P(x)}.\ H_{\neg P(x)}\ (H_{\forall x.\ P(x)}\ x)))
\end{array}
$$

Figure 5. Constructive instances of the Law of Double Negation

5. Inspecting the proof

Seen as a function symbol in type theory, the symbol ldn is a function of two parameters; the first one is a formula A, 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 $\neg A$ 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 canonical constructive proof of $A \Rightarrow \neg\neg A$ (which is $\lambda H_A.\ \lambda H_{\neg A}.\ H_{\neg A}\ H_A$)

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

$$
\begin{array}{ll}
\mathsf{ldn}(A, \lambda H_{\neg A}.\ H_\bot) \hookrightarrow \bot_E^A(H_\bot) & (R_1) \\
\mathsf{ldn}(A, \lambda H_{\neg A}.\ H_{\neg A}\ H_A) \hookrightarrow H_A & (R_2)
\end{array}
$$

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

$$
\mathsf{ldn}(A, \lambda H_{\neg A}.\ (H_{\neg A\Rightarrow\neg A}\ H_{\neg A})\ H_A) \hookrightarrow H_A \quad (R_{2'})
$$

These rules restrict the positions in which $H_{\neg A}$ 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:

$$
\bot_E^A(H_{\neg A}\ H_A) \hookrightarrow H_A \quad (R_3)
$$

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

$$
\begin{array}{ll}
\bot_E^{A\Rightarrow B}(H_\bot)\ H_A \hookrightarrow \bot_E^B(H_\bot) & (R_{\mathsf{abort}-@}) \\
\lambda H_A.\bot_E^B(H_\bot) \hookrightarrow \bot_E^{A\Rightarrow B}(H_\bot) & (R_{\mathsf{abort}-\lambda})
\end{array}
$$

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

Another option for eliminating ldn 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 $H_\bot$ in R_1 and the pattern $H_{\neg A}\ H_A$ in R_2 by looking at where it does use the hypothesis $H_{\neg A}$. $\bot$ has no introduction rule so it can only be proved by an elimination rule. Elimination rules for disjunction and existential can be traversed by ldn if the required proof of $B \vee C$ and $\exists x, P(x)$ respectively do not use the assumption $H_{\neg A}$:

$$\begin{array}{l}
\mathsf{ldn}(A, \lambda H_{\neg A}.\ \vee_E^{\bot}(H_{\neg A \Rightarrow \neg B}\ H_{\neg A}, H_{\neg A \Rightarrow \neg C}\ H_{\neg A}, H_{B \vee C})) \hookrightarrow \\
\quad \vee_E^{A}(\lambda H_B.\ \mathsf{ldn}(A, \lambda H_{\neg A}.\ H_{\neg A \Rightarrow \neg B}\ H_{\neg A}\ H_B), \\
\qquad \lambda H_C.\ \mathsf{ldn}(A, \lambda H_{\neg A}.\ H_{\neg A \Rightarrow \neg C}\ H_{\neg A}\ H_C), \\
\qquad H_{B \vee C}) \\
\mathsf{ldn}(A, \lambda H_{\neg A}.\ \exists_E^{\bot}(H_{\neg A \Rightarrow \forall x. \neg P(x)}\ H_{\neg A}, H_{\exists x. P(x)})) \hookrightarrow \\
\quad \exists_E^{A}(\lambda x. \lambda H_{P(x)}. \mathsf{ldn}(A, \lambda H_{\neg A}.\ H_{\neg A \Rightarrow \forall x. \neg P(x)}\ H_{\neg A}\ x\ H_{P(x)}), \\
\qquad H_{\exists x. P(x)})
\end{array}$$

To ease triggering of these new rules, we want to push elimination rules for disjunction and existential toward the root of the formula in the hope that they will meet the ldn symbol and help it progress toward the leaves of the proof.

We avoid commuting with introduction rules because it goes a lot against cut-elimination and does not seem useful in practice for normal forms with respect to the rewrite system R_{ldn}. Commuting with other elimination rules is however achieved easily:

$$\begin{array}{l}
\bot_E^{C}(\vee_E^{\bot}(H_{\neg A}, H_{\neg B}, H_{A \vee B})) \hookrightarrow \\
\quad \vee_E^{C}(\lambda H_A.\ \bot_E^{C}(H_{\neg A} H_A), \ldots, H_{A \vee B}) \\
\mathsf{fst}(\vee_E^{C \wedge D}(H_{A \Rightarrow C \wedge D}, H_{B \Rightarrow C \wedge D}, H_{A \vee B})) \hookrightarrow \\
\quad \vee_E^{C}(\lambda H_A.\ \mathsf{fst}(H_{A \Rightarrow C \wedge D}\ H_A), \ldots, H_{A \vee B}) \\
\mathsf{snd}(\vee_E^{C \wedge D}(H_{A \Rightarrow C \wedge D}, H_{B \Rightarrow C \wedge D}, H_{A \vee B})) \hookrightarrow \\
\quad \vee_E^{D}(\lambda H_A.\ \mathsf{snd}(H_{A \Rightarrow C \wedge D}\ H_A), \ldots, H_{A \vee B}) \\
\vee_E^{C \Rightarrow D}(H_{A \Rightarrow C \Rightarrow D}, H_{B \Rightarrow C \Rightarrow D}, H_{A \vee B})\ H_C \hookrightarrow \\
\quad \vee_E^{D}(\lambda H_A.\ H_{A \Rightarrow C \Rightarrow D}\ H_A\ H_C, \ldots, H_{A \vee B}) \\
\vee_E^{\forall x.\ P(x)}(H_{A \Rightarrow \forall x.\ P(x)}, H_{B \Rightarrow \forall x.\ P(x)}, H_{A \vee B})\ t \hookrightarrow \\
\quad \vee_E^{P(t)}(\lambda H_A.\ H_{A \Rightarrow \forall x.\ P(x)}\ H_A\ t, \ldots, H_{A \vee B})
\end{array}$$

$$\begin{array}{l}
\bot_E^{A}(\exists_E^{\bot}(H_{\forall x.\ \neg P(x)}, H_{\exists x.\ P(x)})) \hookrightarrow \\
\quad \exists_E^{A}(\lambda x.\ \lambda H_{P(x)}.\ \bot_E^{A}(H_{\forall x.\ \neg P(x)}\ x\ H_{P(x)}), H_{\exists x.\ P(x)}) \\
\mathsf{fst}(\exists_E^{A \wedge B}(H_{\forall x.\ P(x) \Rightarrow A \wedge B}, H_{\exists x.\ P(x)})) \hookrightarrow \\
\quad \exists_E^{A}(\lambda x.\ \lambda H_{P(x)}.\ \mathsf{fst}(H_{\forall x.\ P(x) \Rightarrow A \wedge B}\ x\ H_{P(x)}), \\
\qquad H_{\exists x.\ P(x)}) \\
\mathsf{snd}(\exists_E^{A \wedge B}(H_{\forall x.\ P(x) \Rightarrow A \wedge B}, H_{\exists x.\ P(x)})) \hookrightarrow \ldots \\
\exists_E^{A \Rightarrow B}(H_{\forall x. P(x) \Rightarrow A \Rightarrow B}, H_{\exists x.\ P(x)})\ H_A \hookrightarrow \\
\quad \exists_E^{B}(\lambda x.\ \lambda H_{P(x)}.\ H_{\forall x. P(x) \Rightarrow C \Rightarrow D}\ x\ H_{P(x)}\ H_C, \\
\qquad H_{\exists x.\ P(x)}) \\
\exists_E^{\forall y.\ Q(y)}(H_{\forall x.\ P(x) \Rightarrow \forall y.\ Q(y)}, H_{\exists x.\ P(x)})\ t \hookrightarrow \\
\quad \exists_E^{Q(t)}(\lambda x.\ \lambda H_{P(x)}.\ H_{\forall x.\ P(x) \Rightarrow \forall y.\ Q(y)}\ x\ H_{P(x)}\ t, \\
\qquad H_{\exists x.\ P(x)})
\end{array}$$

5.4 Confluence

The rules R_1 and R_2 are not confluent with the rewrite system R_{ldn} of Section 4. For example, the term $\lambda H_\bot.\mathsf{ldn}(\top, \lambda H_{\neg\top}.\ H_\bot)$ reduces to $\lambda H_\bot.\ \lambda H_{\neg\neg\top}.\ \top_I$ with respect to R_{ldn} and to $\lambda H_\bot.\ \bot_E^{\top}(H_\bot)$ with respect to R_1. Even worse, the rules of Subsection 5.2 are to be used together but they form a non-confluent rewrite system: the term $\bot_E^{A \Rightarrow B}(\pi_{\neg(A \Rightarrow B)} \pi_{A \Rightarrow B})\ \pi_A$ reduces to both $\pi_{A \Rightarrow B}\ \pi_A$ (using R_3) and $\bot_E^{B}(\pi_{\neg(A \Rightarrow B)}\ \pi_{A \Rightarrow B})$ (using $R_{\mathsf{abort}-@}$).

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

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 $\hookrightarrow^*_{R_A \cup R_B}$ by writing both systems in the same file
- sequence: by calling Dedukti twice, we can reduce terms using the relation $\hookrightarrow^*_{R_A} \cdot \hookrightarrow^*_{R_B}$, 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 $R_{\mathsf{abort}-@}$ and $R_{\mathsf{abort}-\lambda}$, 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_{ldn}), then rules pushing the axioms toward the leaves at the expanse 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 ldn 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 $A \Rightarrow A$ that comes out of Zenon is the following term:

$$\begin{array}{l}
\mathsf{ldn}(A \Rightarrow A, \\
\quad \lambda H_{\neg(A \Rightarrow A)}.H_{\neg(A \Rightarrow A)} \\
\qquad (\lambda H_A.\ \bot_E^{A}((\lambda H'_A.\ H_{\neg(A \Rightarrow A)}\ (\lambda H''_A.\ H'_A))\ H_A)))
\end{array}$$

Two rules apply here, the rule for implication from system R_{ldn} and the rule R_3 for elimination of $\bot_E$. Following the heuristic strategy of Section 6, the first step ($R_1 \cup R_2$) is skipped and we apply the rule in R_{ldn} leading to the following term:

$$\begin{array}{l}
d_3(\mathsf{ldn}(A), \lambda H_{\neg(A \Rightarrow A)}.\ H_{\neg(A \Rightarrow A)} \\
\qquad (\lambda H_A.\ \bot_E^{A}((\lambda H'_A.\ H_{\neg(A \Rightarrow A)}\ (\lambda H''_A.\ H'_A))\ H_A)))
\end{array}$$

we now need to unfold the definition of d_3:

$$\begin{array}{l}
d_3(H_{\mathsf{LDN}(B)}, H_{\neg\neg(A \Rightarrow B)}) := \lambda H_A.\ H_{\mathsf{LDN}(B)}\ (\lambda H_{\neg B}. \\
\qquad H_{\neg\neg(A \Rightarrow B)}\ (\lambda H_{A \Rightarrow B}.\ H_{\neg B}\ (H_{A \Rightarrow B}\ H_A)))
\end{array}$$

so our proof of $A \Rightarrow A$ is now

$$\begin{array}{l}
\lambda H_A.\mathsf{ldn}(A, \lambda H_{\neg A}. \\
\quad (\lambda H_{\neg(A \Rightarrow A)}.\ H_{\neg(A \Rightarrow A)}\ (\lambda H'''_A. \\
\qquad \bot_E^{A}((\lambda H'_A.\ H_{\neg(A \Rightarrow A)}\ (\lambda H''_A.\ H'_A))\ H'''_A))) \\
\quad (\lambda H_{A \Rightarrow A}.\ H_{\neg A}\ (H_{A \Rightarrow A}\ H_A))
\end{array}$$

we now perform elimination of $\bot_E$ (R_3), which has priority over β-reduction:

$$\begin{array}{l}
\lambda H_A.\mathsf{ldn}(A, \lambda H_{\neg A}. \\
\quad (\lambda H_{\neg(A \Rightarrow A)}.\ H_{\neg(A \Rightarrow A)}\ (\lambda H'''_A.\ H'''_A)) \\
\quad (\lambda H_{A \Rightarrow A}.\ H_{\neg A}\ (H_{A \Rightarrow A}\ H_A))
\end{array}$$

we now β-reduce:

$$\lambda H_A.\mathsf{ldn}(A, \lambda H_{\neg A}.\ H_{\neg A}\ H_A)$$

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 β-redex $(\lambda H'_A.\ H_{\neg(A\Rightarrow A)}\ (\lambda H''_A.\ H'_A))\ H_A$. Reducing this cut would forbid to fire R_3 but we need β-reduction to simplify the resulting proof so that R_2 can in turn be fired.

8. Experimental results

We have performed tests on the latest 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 with a short timeout[1]. Zenon claimed to have proved 1371 problems which form our starting benchmark. For every problems in this benchmark but two, Zenon provided a proof in Classical Sequent Calculus that we typechecked in Dedukti. Among these 1369 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. We can distinguish four sources of failure:

1. normalization reaches memory or time limits because matching of higher-order patterns can be costly;
2. some TPTP problems are classical theorems but have no constructive proofs;
3. some problems have constructive proofs but these proofs require ingredients that are not present in the classical proof provided by Zenon, a typical example would be a formula of the form $\varphi \vee P \vee \neg P$ where φ has a complex intuitionistic proof, finding such proofs would require intuitionistic proof search and is out of the scope of our approach;
4. because our approach is heuristic, it is fundamentally incomplete so other proofs are missed, these problems are a good source of inspiration for further improving our heuristics.

The first source of failure affects only 21 proofs (1.7% of the proofs in classical natural deduction). The second source is very hard to count: one goal of the ILTP library (Raths et al. 2005) was to associate a constructive status to TPTP problems but the majority of them (69.7% for ILTP v1.1) remains unsolved or open. Finally, when an intuitionistically valid problem fails to be constructivized by our approach, it is not always clear whether the failure comes from the third or the fourth source because we did not formalize the notion of ingredient present in a classical proof; for example, the formula $P \Rightarrow (P \vee \neg P)$ has two classical proofs, a constructive one and a non-constructive one, our technique fails to constructivize the non-constructive one $\lambda H_A.\ \mathsf{lem}(A)$ as it requires to query the proof context, an operation which can be seen as a very limited form of proof search.

[1] The choice of this timeout does not affect much the results because the number of proofs found by Zenon in more than a few seconds is very low. It has however a direct impact on the time needed for running the benchmark since this timeout is reached on most TPTP problems.

	LK	**NK**	**Normalized**	**NJ**
AGT	17	17	17	13
ALG	23	13	11	7
CAT	2	2	2	2
COM	11	11	11	11
CSR	91	91	87	57
GEO	213	210	210	184
GRA	3	2	2	2
GRP	4	4	4	3
HWV	3	3	3	0
KLE	6	6	6	2
KRS	62	62	62	12
LAT	9	9	8	4
LCL	28	8	8	6
MED	4	4	4	3
MGT	39	38	35	22
MSC	5	5	5	3
NLP	11	11	11	6
NUM	101	92	92	78
PUZ	11	11	10	6
RNG	24	24	24	16
SCT	7	7	7	6
SET	135	135	135	102
SEU	84	80	78	60
SWB	21	21	21	20
SWC	43	43	43	1
SWV	132	132	131	75
SWW	11	11	11	10
SYN	264	201	194	62
SYO	2	2	2	2
TOP	3	3	3	3
Total	1369	1258	1237	778

Each line is a TPTP category of problems;

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

Figure 6. Per-category results

The details for each TPTP category of problem are summarized in Figure 6.

The experimental conditions for this study were the following:

- Processor: Intel Core i5-4310M @ 2.70GHz
- Timeouts: 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/dedukti/dedukti.git` for Dedukti and `git://scm.gforge.inria.fr/zenon/zenon.git` for Zenon). More precisely, Dedukti was built from branch `develop` (latest commit: April 11th 2016), Zenon was built from branch `modulo_intuit` (latest commit: February 5th 2016).

9. Related Work

The differences between classical and intuitionistic logic have been deeply studied since the early days 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, iLeanCoP in particular, 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_{ldn} is related to the way one can replace a double-negation translation by another one. Because the right-hand side of the rule for $\mathsf{ldn}(A \Rightarrow B)$ uses $\mathsf{ldn}(B)$ but not $\mathsf{ldn}(A)$, the correct way to look at R_{ldn} is as a transformer for polarized double-negation translations (Boudard and Hermant 2013).

In the particular case of Zenon proofs in Classical Sequent Calculus and their translation to Classical Natural Deduction, the Law of Double Negation 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 normalization by our rewrite system R_{ldn}, 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 (Raths et al. 2005) is a benchmark constructed from the TPTP problems in FOF 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 as a constructivization extension of a classical prover, LeanCoP.

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 $A \Rightarrow (A \vee \neg A)$: $\lambda H_A.\, \mathsf{lem}(A)$. 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 $A \Rightarrow A$ 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 systems 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 $((A \Rightarrow B) \Rightarrow A) \Rightarrow A$ which is a possible type for the call-cc control operator found in the Scheme programming language for example. This remark lead to Parigot's $\lambda\mu$-calculus which corresponds to 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 $\bar{\lambda}\mu\tilde{\mu}$-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 λ-calculus, it is possible to ask in these systems whether a given term (that is, a classical proof) reduces to a pure λ-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. We expect these proofs to be particularly easy to constructivize because the user is often encouraged in this context to define decidable predicates as functions to type **bool** and $\forall b : \mathbf{bool}.\ b \vee \neg b$ is constructively provable so even the very simple rewrite system R_{lem} together with a rewrite rule 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 $\neg A \vee \neg\neg A$ 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

M. Boudard and O. Hermant. Polarizing double negation translations. *CoRR*, abs/1312.5420, 2013. URL `http://arxiv.org/abs/1312.5420`.

G. Burel. A Shallow Embedding of Resolution and Superposition Proofs into the λΠ-Calculus Modulo. In J. C. Blanchette and J. Urban, editors, *PxTP 2013. 3rd International Workshop on Proof Exchange for Theorem Proving*, volume 14 of *EasyChair Proceedings in Computing*, pages 43–57, Lake Placid, USA, June 2013.

R. Cauderlier and P. Halmagrand. Checking Zenon Modulo Proofs in Dedukti. In Kaliszyk, Cezary and Paskevich, Andrei, editor, Proceedings 4th Workshop on *Proof eXchange for Theorem Proving,* Berlin, Germany, August 2-3, 2015, volume 186 of *Electronic Proceedings in Theoretical Computer Science*, pages 57–73, Berlin, Germany, August 2015. Open Publishing Association. doi: 10.4204/EPTCS.186.7.

P.-L. Curien and H. Herbelin. The duality of computation. *SIGPLAN Not.*, 35(9):233–243, Sept. 2000. ISSN 0362-1340. doi: 10.1145/357766.351262. URL `http://doi.acm.org/10.1145/357766.351262`.

G. Dowek. On the definition of the classical connectives and quantifiers. In *Why is this a Proof? Festschrift for Luiz Carlos Pereira*. College Publication, 2015.

G. Gentzen. Über das verhältnis zwischen intuitionistischer und klassischer arithmetik. *Archiv für mathematische Logik und Grundlagenforschung*, 16(3):119–132, 1974. ISSN 1432-0665. doi: 10.1007/BF02015371. URL `http://dx.doi.org/10.1007/BF02015371`.

F. Gilbert. A lightweight double-negation translation. In A. Fehnker, A. McIver, G. Sutcliffe, and A. Voronkov, editors, *LPAR-20. 20th International Conferences on Logic for Programming, Artificial Intelligence and Reasoning - Short Presentations*, volume 35 of *EPiC Series in Computing*, pages 81–93. EasyChair, 2015.

K. Gödel. Zur intuitionistischen arithmetik und zahlentheorie. *Ergebnisse eines mathematischen Kolloquiums*, 4(1933):34–38, 1933.

R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, Jan. 1993. ISSN 0004-5411. doi: 10.1145/138027.138060. URL `http://doi.acm.org/10.1145/138027.138060`.

A. N. Kolmogorov. On the principle of the excluded middle. In J. van Heijenoort, editor, *A Source Book in Mathematical Logic, 1879–1931*, pages 414–437. Harvard Univ., 1925.

J.-L. Krivine. Opérateurs de mise en mémoire et traduction de gödel. *Archive for Mathematical Logic*, 30(4):241–267, 1990. ISSN 1432-0665. doi: 10.1007/BF01792986. URL `http://dx.doi.org/10.1007/BF01792986`.

J.-L. Krivine. Typed lambda-calculus in classical Zermelo-Frænkel set theory. *Archive for Mathematical Logic*, 40(3):189–205, 2001. ISSN 1432-0665. doi: 10.1007/s001530000057. URL `http://dx.doi.org/10.1007/s001530000057`.

S. Kuroda. Intuitionistische untersuchungen der formalistischen logik. *Nagoya Math. J.*, 2:35–47, 1951.

D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1:253–281, 1991.

B. Nordstrom, K. Petersson, and J. M. Smith. Programming in martin-lof's type theory.an introduction. In *Number 7 in International series of monographs on computer science*. Oxford University Press, 1989.

J. Otten. Clausal connection-based theorem proving in intuitionistic first-order logic. In B. Beckert, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX 2005, Koblenz, Germany, September 14-17, 2005, Proceedings*, volume 3702 of *Lecture Notes in Computer Science*, pages 245–261. Springer, 2005. ISBN 3-540-28931-3. doi: 10.1007/11554554_19. URL `http://dx.doi.org/10.1007/11554554_19`.

M. Parigot. Lambda-my-calculus: An algorithmic interpretation of classical natural deduction. In A. Voronkov, editor, *Logic Programming and Automated Reasoning,International Conference LPAR'92, St. Petersburg, Russia, July 15-20, 1992, Proceedings*, volume 624 of *Lecture Notes in Computer Science*, pages 190–201. Springer, 1992. ISBN 3-540-55727-X. doi: 10.1007/BFb0013061. URL `http://dx.doi.org/10.1007/BFb0013061`.

T. Raths, J. Otten, and C. Kreitz. The ILTP library: Benchmarking automated theorem provers for intuitionistic logic. In B. Beckert, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX 2005, Koblenz, Germany, September 14-17, 2005, Proceedings*, volume 3702 of *Lecture Notes in Computer Science*, pages 333–337. Springer, 2005. ISBN 3-540-28931-3. doi: 10.1007/11554554_28. URL `http://dx.doi.org/10.1007/11554554_28`.

R. Saillard. *Type Checking in the Lambda-Pi-Calculus Modulo: Theory and Practice*. PhD thesis, École des Mines, 2015.