Classical provers are the best intuitionistic provers *

Gabriel Ebner

TU Wien, Vienna, Austria

Abstract. We describe a new method to constructivize proofs based on Herbrand disjunctions by giving a practically effective algorithm that converts (some) classical first-order proofs into intuitionistic proofs. Together with an automated classical first-order theorem prover such a method yields an (incomplete) automated theorem prover for intuitionistic logic. Our implementation of this prover approach, Slakje, performs competitively on the ILTP benchmark suite for intuitionistic provers: it solves 1674 out of 2670 problems (1290 proofs and 384 claims of non-provability) with Vampire as a backend, including 831 previously unsolved problems.

1 Introduction

Intuitionistic logic is a logic of particular practical importance. Many interactive theorem provers use intuitionistic logic as a foundation, like Coq [4], Agda [7], or Lean [28]. In some foundational frameworks the law of excluded middle is even provably false, such as in homotopy type theory[1] [39]. Automating first-order intuitionistic logic thus has immediate practical applications in these systems.

That intuitionistic proofs are often similar to classical proofs of the same formula is a folklore observation, stated e.g. by Otten [31]. Hence it is reasonable to approach automated theorem proving in intuitionistic logic by adapting proofs from classical theorem provers. This general idea of proof constructivization has recently been described and evaluated by Cauderlier [9] and Gilbert [17]; both transform detailed proofs (natural deduction resp. sequent calculus) using essentially local rewriting operations. However their constructivization procedures are hard to apply to state-of-the-art automated theorem provers as these provers typically do not produce sequent calculus or natural deduction proofs.

Integrations of (classical) first-order theorem provers in higher-order theorem provers—so-called "hammers"—typically use a similar general approach: passing a (sometimes even unsound) translation of the input problem to a classical prover, and then reconstructing the proof in the higher-order system [6,11,19]. In

* Supported by the Vienna Science and Technology Fund (WWTF) project VRG12-004

[1] Considering of course the law of excluded middle for arbitrary types, not just mere propositions.

this framework, proof constructivization uses an unsound translation: one that maps each formula to itself (but reinterprets it in a different logic).

We present a new and different proof constructivization method based on Herbrand disjunctions. Herbrand's theorem [18,8] captures the insight that the classical validity of a quantified formula is characterized by the existence of a tautological finite set of quantifier-free instances. In its simplest case, the validity of a purely existential formula $\exists x\, \varphi(x)$ is characterized by the existence of a tautological disjunction of instances $\varphi(t_1) \vee \cdots \vee \varphi(t_n)$, a Herbrand disjunction. To store such Herbrand disjunctions for general non-prenex formulas, we use an elegant data structure called expansion trees, which also generalize this result to simply-typed higher-order logic in the form of elementary type theory [25].

- We describe a new and effective procedure to constructivize classical proofs into intuitionistic proofs based on Herbrand disjunctions.
- We have implemented the intuitionistic first-order theorem prover Slakje based on this procedure using the GAPT [16] system for proof theory, and show that it performs competitively on the ILTP benchmark suite.
- We show that the prover is complete on a practically relevant class of formulas.
- As a special case of this completeness result, we obtain a proof-theoretic proof of the completeness of the propositional prover in [10].

We start out in Section 2 by giving an overview of the Slakje prover. In the following sections we explain the technical details. Expansion trees, the central data structure to represent classical proofs, are introduced in Section 3. In Section 4 we describe the SAT-based procedure that constructivizes expansion proofs and produces intuitionistic proofs. Key optimizations are discussed in Section 5, and completeness for a large class of problems including Horn problems and purely equational problems is shown in Section 6. Finally, we evaluate the prover on the ILTP benchmark suite in Section 7.

2 Overview of the prover

We consider intuitionistic first-order logic with the connectives $\rightarrow, \wedge, \vee, \bot, \top$ and the quantifiers $\exists, \forall$. The abbreviation $\neg\varphi$ stands for $\varphi \rightarrow \bot$. Let us first give a short overview of the resulting intuitionistic first-order prover. Given an input formula φ, it proceeds in three big phases:

1. Call external prover (e.g. Vampire) with φ
   (if the result is "satisfiable", immediately return "non-theorem")
2. Convert proof output into (classical) expansion proof
3. Produce intuitionistic proof from expansion proof

The only phase that is specific to this prover is the third one; phases 1 and 2 are part of the general-purpose external prover interface that produces expansion proofs available in the GAPT [16,13] framework.

We use expansion proofs as a compact intermediate format for classical proofs, which only contain the quantifier instances but abstract from the propositional reasoning in the proof. Many automated theorem proving paradigms generate proofs that contain the same essential data as expansion proofs, e.g. the terms used for heuristic instantiation in SMT solvers, or the global substitution used in tableaux or connection proofs. Resolution and superposition proofs contain this information after grounding. This direct correspondence applies for formulas in clause normal form; in the general case we also need to treat strong quantifiers, which are Skolemized in classical provers.

While there are normal forms similar to CNF in intuitionistic logic which avoid Skolemization [26,27], it makes little sense to use them here: the main difference is that they produce a different kind of "clauses", such as $(\forall x\, P(x,y)) \to Q(y)$, which we cannot pass to classical theorem provers. But the use of Skolemization as a preprocessing step is (in general) not sound in intuitionistic logic: for example $(\neg\forall x\ P(x)) \to \exists x\ \neg P(x)$ is an intuitionistic non-theorem, while its Skolemization $(\neg P(c)) \to \exists x\ \neg P(x)$ is a theorem. Hence we use *deskolemization* [2] to eliminate Skolemization from classical proofs (which is naturally defined on expansion proofs). Using an additional preprocessing step, this procedure can be extended to proofs containing equational reasoning [1].

The way we construct an intuitionistic proof from the expansion proof is by a bottom-up proof construction in an intuitionistic multi-succedent sequent calculus. We make use of a SAT solver to organize this proof construction. While SAT solvers—as the name implies—can decide satisfiability of propositional formulas in classical logic, only the clausification of the formula requires classical logic. The proofs produced by SAT solvers can be translated to resolution proofs; and resolution is just the cut inference in our calculus, which is sound for intuitionistic logic. In our setting, a SAT solver hence decides the following question: "is the sequent $\mathcal{S}$ derivable from a set of sequents $\mathcal{T}$ using only cut and weakening?"

If the SAT solver cannot derive this sequent, we obtain an assignment which corresponds to a leaf in this bottom-up proof construction and we apply the inferences that cannot be encoded as clauses (e.g. the right-rule for implication). This technique of using SAT solvers to support intuitionistic reasoning has already been successfully used in the Intuit [10] prover, albeit only for propositional logic, and their implementation does not produce proofs.

3 Expansion proofs

The proof formalism of expansion trees was introduced in [25] to describe Herbrand disjunctions in *classical* higher-order logic. In first-order logic, they provide an elegant data structure to describe Herbrand disjunctions for non-prenex formulas. The main data stored in expansion proofs are the quantifier instance terms. The central idea is that each expansion tree E comes with a *shallow formula* $\mathrm{sh}(E)$ and a quantifier-free *deep formula* $\mathrm{dp}(E)$. The deep formula corresponds to the quantifier-free Herbrand disjunction, and the shallow formula is the quantified formula that we want to prove. If the deep formula is a quasi-

tautology (a tautology modulo equality), then the shallow formula is valid in classical logic.

Example 1. Consider the formula[2] $\varphi := \forall x\, P(x) \to (\forall x\, P(f(x)) \to Q) \to Q$. The expansion tree $E := (\forall x\, P(x) +^{f(\alpha)} P(f(\alpha))) \to (\forall x\, P(f(x)) +^{\alpha}_{\text{ev}} P(f(\alpha)) \to Q) \to Q$ has the shallow formula $\text{sh}(E) = \varphi$, and its deep formula $\text{dp}(E) = (P(f(\alpha)) \to (P(f(\alpha)) \to Q) \to Q)$ is tautological. The quantifier instance terms here are $f(\alpha)$ and α (written in superscript after the $+$).

An instructive way to think about expansion proofs is that they are a compressed form of cut-free sequent calculus proofs where we only store the quantifier inferences. The following proof uses exactly the same terms, $f(\alpha)$ and α, in the quantifier inferences $\forall_l$ and $\forall_r$, resp:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P(f(\alpha)) \vdash P(f(\alpha))}{\forall x\, P(x) \vdash P(f(\alpha))}\,\forall_l}{\forall x\, P(x) \vdash \forall x\, P(f(x))}\,\forall_r \qquad Q \vdash Q}{\forall x\, P(x), \forall x\, P(f(x)) \to Q \vdash Q}\,\to_l}{\forall x\, P(x) \vdash (\forall x\, P(f(x)) \to Q) \to Q}\,\to_r}{\vdash \forall x\, P(x) \to (\forall x\, P(f(x)) \to Q) \to Q}\,\to_r$$

Expansion trees have two polarities: in Example 1 above, the first $\forall$ has negative polarity and the second $\forall$ has positive polarity. Polarity only changes on the left side of the connective $\to$. (In the sequent calculus proof we used $\forall_l$ for the negative and $\forall_r$ for the positive polarity.) This distinction is important since we must instantiate the second one with a variable, while we can instantiate the first one with whatever terms we want.

Definition 1. *The set* $\text{ET}^p(\varphi)$ *of* expansion trees *with polarity* $p \in \{+, -\}$ *and* shallow formula φ *is inductively defined as the smallest set containing:*

$$\frac{A \text{ atom}/\top/\bot}{A^p \in \text{ET}^p(A)} \qquad \frac{E_1 \in \text{ET}^p(\varphi) \qquad E_2 \in \text{ET}^p(\psi)}{E_1 \wedge E_2 \in \text{ET}^p(\varphi \wedge \psi)}$$

$$\frac{E_1 \in \text{ET}^p(\varphi) \qquad E_2 \in \text{ET}^p(\psi)}{E_1 \vee E_2 \in \text{ET}^p(\varphi \vee \psi)} \qquad \frac{E_1 \in \text{ET}^{-p}(\varphi) \qquad E_2 \in \text{ET}^p(\psi)}{E_1 \to E_2 \in \text{ET}^p(\varphi \to \psi)}$$

$$\frac{E \in \text{ET}^+(\varphi[x\backslash y])}{\forall x\, \varphi +^{y}_{\text{ev}} E \in \text{ET}^+(\forall x\, \varphi)} \qquad \frac{E_1 \in \text{ET}^-(\varphi[x\backslash t_1]) \quad \cdots \quad E_n \in \text{ET}^-(\varphi[x\backslash t_n])}{\forall x\, \varphi +^{t_1} E_1 \cdots +^{t_n} E_n \;\in\; \text{ET}^-(\forall x\, \varphi)}$$

$$\frac{E \in \text{ET}^-(\varphi[x\backslash y])}{\exists x\, \varphi +^{y}_{\text{ev}} E \in \text{ET}^-(\exists x\, \varphi)} \qquad \frac{E_1 \in \text{ET}^+(\varphi[x\backslash t_1]) \quad \cdots \quad E_n \in \text{ET}^+(\varphi[x\backslash t_n])}{\exists x\, \varphi +^{t_1} E_1 \cdots +^{t_n} E_n \;\in\; \text{ET}^+(\exists x\, \varphi)}$$

Each expansion tree E has a uniquely determined shallow formula and polarity, we write $\text{sh}(E)$ for its shallow formula, and $\text{pol}(E)$ for its polarity. Given an expansion tree $E = Qx\varphi +^{\alpha}_{\text{ev}} E'$ where $Q \in \{\forall, \exists\}$, we say that α is the

[2] We use the convention that the quantifiers $\forall, \exists$ bind stronger than $\to, \wedge, \vee$. That is, $\forall x\, P(x) \to Q$ is the same formula as $(\forall x\, P(x)) \to Q$.

eigenvariable of E. While the shallow formula describes the quantified formula to be proven, the deep formula is a quantifier-free formula corresponding to the Herbrand disjunction:

Definition 2. *Let E be an expansion tree, we define the* deep formula $\mathrm{dp}(E)$ *recursively as follows:*

$$\mathrm{dp}(A^p) = A, \quad \mathrm{dp}(\top^p) = \top, \quad \mathrm{dp}(\bot^p) = \bot, \quad \mathrm{dp}(E_1 \wedge E_2) = \mathrm{dp}(E_1) \wedge \mathrm{dp}(E_2)$$

$$\mathrm{dp}(E_1 \vee E_2) = \mathrm{dp}(E_1) \vee \mathrm{dp}(E_2), \quad \mathrm{dp}(E_1 \to E_2) = \mathrm{dp}(E_1) \to \mathrm{dp}(E_2)$$

$$\mathrm{dp}(\forall x\, \varphi +^{y}_{\mathrm{ev}} E) = \mathrm{dp}(E), \quad \mathrm{dp}(\forall x\, \varphi +^{t_1} E_1 \cdots +^{t_n} E_n) = \mathrm{dp}(E_1) \wedge \cdots \wedge \mathrm{dp}(E_n)$$

$$\mathrm{dp}(\exists x\, \varphi +^{y}_{\mathrm{ev}} E) = \mathrm{dp}(E), \quad \mathrm{dp}(\exists x\, \varphi +^{t_1} E_1 \cdots +^{t_n} E_n) = \mathrm{dp}(E_1) \vee \cdots \vee \mathrm{dp}(E_n)$$

In an expansion proof the eigenvariables need to be acyclic. This criterion is a similar restriction to the eigenvariable condition in sequent calculi. Formally we require the following dependency relation to be acyclic:

Definition 3. *Let E be an expansion tree. The dependency relation $<_E$ is a binary relation on variables where $x <_E y$ iff E contains a subtree E' such that $x \in \mathrm{FV}(\mathrm{sh}(E'))$ and y is an eigenvariable in of a subtree of E'.*

Definition 4. *An* expansion proof *E of φ is an $E \in \mathrm{ET}^+(\varphi)$ such that:*

1. *$<_{\mathcal{E}}$ is acyclic (i.e., can be extended to a linear order) and there are no duplicate eigenvariables, and*
2. *$\mathrm{dp}(\mathcal{E})$ is a quasi-tautology*

Theorem 1. *A formula φ is a theorem of classical first-order logic if and only if there exists an expansion proof $E \in \mathrm{ET}^+(\varphi)$.*

Example 2. The formula $\exists x\, p(c) \to p(x)$ has $E_1 = \exists x\, p(c) \to p(x) +_c (p(c)^- \to p(c)^+)$ as an expansion proof. The deep formula $\mathrm{dp}(E_1) = p(c) \to p(c)$ is a tautology.

Expansion proofs are closely related to the matrix characterization for classical first-order logic [5] used by connection-based theorem provers [32]. Both separate the proof into two layers: the quantifier inference terms, and the propositional proof. In connection proofs, the quantifier instance terms are stored implicitly as the result of the unifier induced by the connections, while expansion proofs contain these terms explicitly. The multiplicity in a connection proof corresponds essentially to the number of children in the weak quantifier nodes of an expansion tree. Integrating equality into connection proofs is hard as it requires simultaneous rigid E-unification [40], and connection provers such as leanCoP hence opt to add axioms for reflexivity, transitivity, and congruence of equality during preprocessing.

By contrast, expansion proofs work modulo equality. We do not need add explicit axioms for equality. Instead, the handling of equality is part of verifying that the deep formula is a quasi-tautology, and can be done using off-the-shelf SMT solvers. In principle, this could also be extended to other decidable (and for our purposes, intuitionistically valid) theories used in SMT solvers such as Presburger arithmetic.

4 Proof constructivization

Our proof constructivization method operates on the level of expansion proofs. That is, it takes an expansion proof and (if successful) produces an intuitionistic proof using (at most) the quantifier inferences indicated by the expansion proof. Formally, we want to find a proof in the multi-succedent intuitionistic sequent calculus mLJ as shown in Fig. 1, where all eigenvariables and weak quantifier instances occur in the expansion proof and there are no duplicate eigenvariables along any branch of the proof.

Definition 5. *A proof π of a sequent $\mathcal{S}$ in mLJ* realizes *an expansion proof E iff $\mathcal{S} = \mathrm{sh}(E)$ and every quantifier instance term in π is contained in E, i.e.:*

- *If* $\dfrac{\Gamma \vdash \varphi(x)}{\Gamma \vdash \forall x\, \varphi(x)}$ *is a subproof of π, then $\forall x\, \varphi(x) +^{x}_{\mathrm{ev}} E'$ is a subtree of E (for some E')*
- *If* $\dfrac{\varphi(t), \Gamma \vdash \Delta}{\forall x\, \varphi(x), \Gamma \vdash \Delta}$ *is a subproof of π, then $\forall x\, \varphi(x) +^{t} E' \cdots$ is a subtree of E (for some E')*

(and analogously for $\exists$)

Note that Definition 5 ignores the ancestor relationship of formulas in a proof: if a (sub-)formula occurs twice in the expansion proof, then the instances for one occurrence can be used for the other as well and vice versa.

Definition 6. *A proof π in mLJ is called* weakly regular *iff for all subproofs of π of the form* $\dfrac{\Gamma \vdash \varphi(\alpha)^{(\psi)}}{\Gamma \vdash \forall x\, \varphi(x)}\,\forall_r$ *or* $\dfrac{\varphi(\alpha)^{(\psi)}, \Gamma \vdash \Delta}{\exists x\, \varphi(x), \Gamma \vdash \Delta}\,\exists_l$*, the eigenvariable α does not occur as the eigenvariable of an inference in ψ.*

Our algorithm will have the property that whenever a cut-free weakly regular proof exists which realizes E, the algorithm will succeed and return an intuitionistic proof. The restriction of cut-free weak regularity is due to the intuitionistic logic; in classical logic, we can always find a cut-free weakly regular proof realizing E, provided that $\mathrm{dp}(E)$ is quasi-tautological.

Example 3. Consider the expansion proof $\vdash \neg\neg(\forall x\,(p \vee \neg p) +^{\alpha}_{\mathrm{ev}} (p^{+} \vee \neg p^{-}))$. This expansion proof cannot be realized by a weakly regular cut-free proof, since we would need to use two $\forall_r$ inferences but the expansion proof only contains one eigenvariable. The natural proof would use the eigenvariable α twice:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{p \vdash p}{p \vdash p \vee \neg p}\, w_r, \vee_r}{p \vdash \forall x\,(p \vee \neg p)}\, \forall_r \text{ (using the eigenvariable } \alpha)}{\neg\forall x\,(p \vee \neg p) \vdash p \vee \neg p}\, \neg_l, \neg_r, w_r, \vee_r}{\neg\forall x\,(p \vee \neg p) \vdash \forall x\,(p \vee \neg p)}\, \forall_r \text{ (using the eigenvariable } \alpha)}{\vdash \neg\neg\forall x\,(p \vee \neg p)}\, \neg_l, \neg_r$$

$$\frac{}{\varphi \vdash \varphi}\,\mathrm{ax} \quad \frac{\Gamma \vdash \Delta}{\varphi, \Gamma \vdash \Delta}\,w_l \quad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, \varphi}\,w_r \quad \frac{\Gamma \vdash \Delta, \varphi \qquad \varphi, \Pi \vdash \Lambda}{\Gamma, \Pi \vdash \Delta, \Lambda}\,\mathrm{cut}$$

$$\frac{}{\vdash t = t}\,\mathrm{rfl} \quad \frac{\Gamma \vdash \Delta, \varphi(t)}{\Gamma, t = s \vdash \Delta, \varphi(s)}\,\mathrm{eq}_r^{\rightarrow} \quad \frac{\Gamma \vdash \Delta, \varphi(s)}{\Gamma, t = s \vdash \Delta, \varphi(t)}\,\mathrm{eq}_r^{\leftarrow}$$

$$\frac{\varphi(t), \Gamma \vdash \Delta}{\varphi(s), \Gamma, t = s \vdash \Delta}\,\mathrm{eq}_l^{\rightarrow} \quad \frac{\varphi(s), \Gamma \vdash \Delta}{\varphi(t), \Gamma, t = s \vdash \Delta}\,\mathrm{eq}_l^{\leftarrow}$$

$$\frac{}{\vdash \top}\,\top_r \quad \frac{}{\bot \vdash}\,\bot_l \quad \frac{\Gamma \vdash \Delta, \varphi, \psi}{\Gamma \vdash \Delta, \varphi \vee \psi}\,\vee_r \quad \frac{\varphi, \Gamma \vdash \Delta \qquad \psi, \Gamma \vdash \Delta}{\varphi \vee \psi, \Gamma \vdash \Delta}\,\vee_l$$

$$\frac{\varphi, \psi, \Gamma \vdash \Delta}{\varphi \wedge \psi, \Gamma \vdash \Delta}\,\wedge_l \quad \frac{\Gamma \vdash \Delta, \varphi \qquad \Gamma \vdash \Delta, \psi}{\Gamma \vdash \Delta, \varphi \wedge \psi}\,\wedge_r$$

$$\frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}\,\to_r \quad \frac{\Gamma \vdash \Delta, \varphi \qquad \psi, \Gamma \vdash \Delta}{\varphi \to \psi, \Gamma \vdash \Delta}\,\to_l$$

$$\frac{\Gamma \vdash \Delta, \varphi(t)}{\Gamma \vdash \Delta, \exists x\, \varphi(x)}\,\exists_r \quad \frac{\varphi(\alpha), \Gamma \vdash \Delta}{\exists x\, \varphi(x), \Gamma \vdash \Delta}\,\exists_l \quad \frac{\varphi(t), \Gamma \vdash \Delta}{\forall x\, \varphi(x), \Gamma \vdash \Delta}\,\forall_l \quad \frac{\Gamma \vdash \varphi(\alpha)}{\Gamma \vdash \forall x\, \varphi(x)}\,\forall_r$$

Fig. 1. The multi-succedent calculus mLJ for intuitionistic first-order logic (variant of L'J first introduced by Maehara [22] but using sets instead of sequences, see also mG1i in Troelstra and Schwichtenberg's classification [38]). A sequent $\Gamma \vdash \Delta$ consists of two sets of formulas Γ and Δ and is interpreted as the formula $\bigwedge \Gamma \to \bigvee \Delta$. The variable α in the $\exists_l$ and $\forall_r$ inferences is called an eigenvariable, and may not occur in Γ, Δ as a free variable. The proof system is cut-free complete for intuitionistic first-order logic with equality.

The SAT-based bottom-of proof construction is done in the Construct and Solve procedures shown in Algorithm 1. The main function Solve applies the inference rules $\exists_l, \forall_r$, and $\to_r$ and calls itself recursively with the premise of these inferences. However it does this in a loop where it first extends the given sequent to a maximal sequent (corresponding to the model returned by the SAT solver). Such a sequent corresponds to a leaf in a restricted bottom-up search, which only uses inferences that we have encoded as clauses in the SAT solver. These clauses are asserted at the beginning of Construct.

We use a standard interface to the SAT solver: the function Assert adds a clause, and IsSatisfiable($\mathcal{S}$) checks whether the clauses are satisfiable given the assumptions $\mathcal{S}$. The function IsESatisfiable runs congruence closure after IsSatisfiable and checks whether the clauses are satisfiable modulo equality.

Concretely we associate to every shallow formula φ of a subtree of E a variable $\|\varphi\|$ in the SAT solver. We implicitly convert the clauses and models of the SAT solver to sequents. Consider the sequent $\mathcal{S} = \varphi_1, \ldots, \varphi_n \vdash \psi_1, \ldots, \psi_m$. Then Assert($\mathcal{S}$) is equivalent to Assert($-\|\varphi_1\|, \ldots, -\|\varphi_n\|, \|\psi_1\|, \ldots, \|\psi_m\|$). The other functions use the opposite polarity: IsESatisfiable($\mathcal{S}$) is equivalent to IsESatisfiable($\|\varphi_1\|, \ldots, \|\varphi_n\|, -\|\psi_1\|, \ldots, -\|\psi_m\|$); the same convention also applies to GetModel and UnsatCore.

Algorithm 1 terminates, since each recursive call of Solve either increases the size of Σ or increases the size of the antecedent of $\mathcal{S}$ while keeping Σ the same.

The overall structure of alternating invertible phases with a decision-making phase is reminiscent of the asynchronous and synchronous rules in focused calculi [21].

Theorem 2. *If* Construct*(E) returns true, then E can be realized by a proof in mLJ (and* $\mathrm{sh}(E)$ *is an intuitionistic theorem).*

Proof. We store a proof for every sequent that is passed to Assert; these have all straightforward proofs in mLJ. Whenever IsESatisfiable$(\mathcal{S})$ returns false for a sequent $\mathcal{S}$, we have an mLJ-proof of $\mathcal{S}$ by combining the asserted sequents using cuts as in the resolution refutation returned by the SAT solver. □

Let us now prove completeness, i.e., Construct returns true if the expansion proof E is realized by a weakly regular proof π in mLJ. Intuitively, the procedure succeeds because it can just pick the same inferences as in π. In a sense, the function Solve proceeds upwards through the proof π, the SAT solver jumps over all inferences except $\exists_l, \forall_r, \to_r$, and (if successful) each model $\Gamma \vdash \Delta$ that we consider in Solve corresponds to an $\exists_l, \forall_r$ or $\to_r$ inference in π.

Formally, we capture the required properties for the model obtained from the SAT solver as "maximal" sequents. Whenever a proof ends in a sub-sequent of a maximal sequent, we can trace the proof upwards to find a $\exists_l, \forall_r$ or $\to_r$ inference also ending in a sub-sequent of the maximal sequent.

Definition 7. *A sequent $\Gamma \vdash \Delta$ is called maximal iff:*

- *$\Gamma \cup \Delta$ are all shallow formulas of subtrees of the expansion proof E.*
- *$\Gamma \cap \Delta = \emptyset$, $\bot \in \Delta$, $\top \in \Gamma$*
- *If $\varphi \wedge \psi \in \Gamma$, then $\varphi, \psi \in \Gamma$.*
- *If $\varphi \wedge \psi \in \Delta$, then $\varphi \in \Gamma$ or $\psi \in \Gamma$.*
- *If $\varphi \vee \psi \in \Delta$, then $\varphi, \psi \in \Delta$.*
- *If $\varphi \vee \psi \in \Gamma$, then $\varphi \in \Gamma$ or $\psi \in \Gamma$.*
- *If $\varphi \to \psi \in \Gamma$, then $\varphi \in \Delta$ or $\psi \in \Gamma$.*
- *If $\forall x\, \varphi(x) \in \Gamma$ and $\forall x\, \varphi(x) +_{t_1} \cdots +_{t_n}$ is a subtree of E, then $\varphi(t_1), \ldots, \varphi(t_n) \in \Gamma$.*
- *If $\exists x\, \varphi(x) \in \Gamma$ and $\exists x\, \varphi(x) +_{t_1} \cdots +_{t_n}$ is a subtree of E, then $\varphi(t_1), \ldots, \varphi(t_n) \in \Delta$.*

Lemma 1. *The sequent $\Gamma \vdash \Delta$ obtained in* Solve *is always maximal.*

Proof. Because of the clauses asserted in Construct. □

Lemma 2. *Let $\mathcal{S}$ be a maximal sequent, and π be a subproof of an mLJ-proof realizing E whose end-sequent is a subsequent of $\mathcal{S}$. Then there is a subproof π' of π such that the end-sequent of π' is also a sub-sequent of $\mathcal{S}$, and π' ends in a $\exists_l, \forall_r$ or $\to_r$ inference.*

Proof. By straightforward induction on π. For illustration, let us prove the case where π ends in an $\wedge_r$-inference: $\dfrac{\begin{matrix}(\pi_1) \\ \Gamma \vdash \Delta, \varphi\end{matrix} \qquad \begin{matrix}(\pi_2) \\ \Gamma \vdash \Delta, \psi\end{matrix}}{\Gamma \vdash \Delta, \varphi \wedge \psi}\, \wedge_r$

Algorithm 1 SAT-based proof constructivization

procedure CONSTRUCT(E)
  ▷ returns true if we have found an intuitionistic proof of $\mathrm{sh}(E)$
  for all subtrees $E_1 \wedge E_2$ of E **do** ▷ and analogously for $\rightarrow, \vee, \bot, \top$
    ASSERT($\mathrm{sh}(E_1) \wedge \mathrm{sh}(E_2) \vdash \mathrm{sh}(E_1)$)
    ASSERT($\mathrm{sh}(E_1) \wedge \mathrm{sh}(E_2) \vdash \mathrm{sh}(E_2)$)
    ASSERT($\mathrm{sh}(E_1), \mathrm{sh}(E_2) \vdash \mathrm{sh}(E_1) \wedge \mathrm{sh}(E_2)$)
  end for
  for all subtrees $\forall x\, \varphi(x) +^{t_1} E_1 \cdots +^{t_n} E_n$ of E **do** ▷ and analogously for $\exists$
    for $1 \leq i \leq n$ **do**
      ASSERT($\forall x\, \varphi(x) \vdash \varphi(t_i)$)
    end for
  end for
  return SOLVE($\emptyset; \mathrm{sh}(E)$)
end procedure

procedure SOLVE($\Sigma; \mathcal{S}$) ▷ Σ is the set of eigenvariables that we have already used
  ▷ $\mathcal{S}$ is the sequent that we want to prove
  ▷ returns true if we have found an intuitionistic proof of $\mathcal{S}$
  while ISESATISFIABLE($\mathcal{S}$) **do**
    $(\Gamma \vdash \Delta)$ = GETMODEL()
    for all $\varphi \rightarrow \psi$ in Δ such that $\varphi \notin \Gamma$ **do**
      if SOLVE($\Sigma; \Gamma, \varphi \vdash \psi$) **then**
        $\Gamma', \varphi \vdash \psi$ = UNSATCORE($\Gamma, \varphi \vdash \psi$)
        ASSERT($\Gamma' \vdash \varphi \rightarrow \psi$)
        continue outer loop
      end if
    end for
    for all $\forall x\, \varphi(x)$ in Δ such that α is an eigenvariable in E but not yet in Σ **do**
      $\Gamma_\alpha = \{\psi \in \Gamma \mid \alpha \notin \mathrm{FV}(\psi)\}$
      if SOLVE($\Sigma \cup \{\alpha\}; \Gamma_\alpha, \vdash \varphi(\alpha)$) **then**
        $\Gamma' \vdash \varphi(\alpha)$ = UNSATCORE($\Gamma_\alpha \vdash \varphi(\alpha)$)
        ASSERT($\Gamma' \vdash \forall x\, \varphi(x)$)
        continue outer loop
      end if
    end for
    for all $\exists x\, \varphi(x)$ in Γ such that α is an eigenvariable in E but not yet in Σ **do**
      $\Gamma_\alpha = \{\psi \in \Gamma \mid \alpha \notin \mathrm{FV}(\psi)\}$
      $\Delta_\alpha = \{\psi \in \Delta \mid \alpha \notin \mathrm{FV}(\psi)\}$
      if SOLVE($\Sigma \cup \{\alpha\}; \Gamma_\alpha, \varphi(\alpha) \vdash \Delta_\alpha$) **then**
        $\Gamma', \varphi(\alpha) \vdash \Delta'$ = UNSATCORE($\Gamma_\alpha, \varphi(\alpha) \vdash \Delta_\alpha$)
        ASSERT($\Gamma', \exists x\, \varphi(x) \vdash \Delta'$)
        continue outer loop
      end if
    end for
    return false
  end while
  return true
end procedure

Let $\mathcal{S} = (\Gamma' \vdash \Delta')$. Note that $\Gamma \vdash \Delta, \varphi \wedge \psi$ is a subsequent of $\Gamma' \vdash \Delta'$ by assumption and hence $\varphi \wedge \psi \in \Delta'$. The sequent $\mathcal{S}$ is maximal, so $\varphi \in \Delta'$ or $\psi \in \Delta'$. First consider $\varphi \in \Delta'$; then $\Gamma \vdash \Delta, \varphi$ is a subsequent of $\mathcal{S}$ and we can apply the induction hypothesis. The case $\psi \in \Delta'$ is symmetric. □

Theorem 3. *If E can be realized by a weakly regular cut-free proof in mLJ, then* Construct*(E) returns true.*

Proof. Let π be a weakly regular cut-free proof that realizes E. We use the following invariant for Solve$(\Sigma; \mathcal{S})$: if there is a subproof ψ of π such that the end-sequent of ψ is a subsequent of $\mathcal{S}$ and the eigenvariables in ψ are disjoint from Σ, then Solve returns true.

In Solve, let ψ be the subproof described above, and let $\mathcal{S}' \supseteq \mathcal{S}$ be the model obtained in the loop. Then $\mathcal{S}'$ is a maximal sequent and there is a subproof ψ' of ψ whose end-sequent is a sub-sequent of $\mathcal{S}'$ as well. The subproof ψ' ends in a $\exists_l, \forall_r$ or $\to_r$ inference. At least one of the recursive calls then corresponds to this inference, and invokes Solve with the premise of the inference. Weak regularity of ψ ensures that the precondition for Σ is satisfied. □

For quantifier-free formulas, the constructivization method is a decision procedure since mLJ is cut-free complete and proofs of quantifier-free formulas are trivially weakly regular and realize the expansion proof. The following corollary also gives a proof-theoretic proof of the completeness of the algorithm used in Intuit prover [10] (their paper gives a proof based on Kripke models).

Corollary 1. *If* sh(E) *is a quantifier-free formula, then* Construct*(E) returns true if and only if* sh(E) *is an intuitionistic theorem.*

5 Optimizations

For performance reasons, we implement several optimizations in the Solve procedure. The first one was already described in [10].

Caching unsolvable cases. By using a SAT solver, we already have a cache for the solvable cases: whenever Solve$(\Sigma; \mathcal{S})$ returns true, the SAT solver remembers the conflict clause and all subsequent calls to Solve will terminate after just one call to IsESatisfiable. However if we cannot find a proof, then we would need to repeat the costly recursive backtracking procedure. Hence we store all pairs $(\Sigma; \Gamma \vdash \Delta)$ where the result is false ($\Gamma \vdash \Delta$ is the model obtained in Solve which extends $\mathcal{S}$). At the beginning of Solve we check if we have already stored a pair $(\Sigma'; \mathcal{S}')$ such that $\Sigma \subseteq \Sigma'$ and $\mathcal{S} \supseteq \mathcal{S}'$, and return false if there is such a pair. (We use a trie-like data structure to store these pairs.)

Classical quasi-tautology check. If a sequent $\Gamma \vdash \Delta$ is not even classically provable, then it cannot be intuitionistically provable either. This easy observation allows us to prune large branches of the backtracking search in Solve. Concretely, we add an atom `classical`. For implication, we assert all sequents

$\texttt{classical} \vdash \mathrm{sh}(E_1) \to \mathrm{sh}(E_2), \mathrm{sh}(E_1)$ where $E_1 \to E_2$ is a subtree of the expansion proof E. For universal quantification we assert $\texttt{classical}, \varphi(\alpha) \vdash \forall x\, \varphi(x)$ for all subtrees $\forall x\, \varphi(x) +^{\alpha}_{\mathrm{ev}} \cdots$. And for existential quantification, we assert $\texttt{classical}, \exists x\, \varphi(x) \vdash \varphi(\alpha)$ for all subtrees $\exists x\, \varphi(x) +^{\alpha}_{\mathrm{ev}} \cdots$. At the beginning of the Solve procedure, we call IsESatisfiable($\texttt{classical}, \Gamma \vdash \Delta$), and immediately return false if IsESatisfiable returns true.

Invertible occurrences of $\exists_l$. In some cases we can avoid backtracking with existential quantifiers in the antecedent. This is the case if we have a subtree $\exists x\, \varphi(x) +^{\alpha}_{\mathrm{ev}} E_1$, where all free variables in $\exists x\, \varphi(x)$ are already in Σ. In this case we immediately apply the corresponding $\exists_l$ inference, and skip all the loops in the Solve procedure. This is correct because we can permute the $\exists_l$ inference downward in the realizing proof.

6 Completeness on subclasses

In general, our proof constructivization-based approach to intuitionistic theorem proving is incomplete. For example, $\forall x\, (p(x) \vee \neg p(x)) \vdash \neg\neg p(c) \to p(c)$ is an intuitionistic theorem where our approach will fail—we clearly need the assumption $\forall x\, (p(x) \vee \neg p(x))$, but virtually all classical theorem provers will discard it immediately and never use it. For decidability assumptions such as $\forall x\, (p(x) \vee \neg p(x))$ we use heuristic instantiation as a post-processing step, adding all instances of the formula for subterms occurring in the expansion proof.

However there are some classes of formulas where our approach is complete. These are classes of first-order formulas where intuitionistic provability is equivalent to classical provability, such classes are called Glivenko classes and were e.g. studied by Orevkov. See also [29] for a more modern presentation.

Definition 8 ([30]). Class 1 *is the set of sequents which do not have positive occurrences of $\to$ or $\forall$.*

Example 4. The sequent $\forall x\, (p(x) \to q(x) \vee \neg r(x)), p(c) \wedge r(c) \vdash q(c)$ is in Class 1; $(p \to q) \to p \vdash$ and $(\forall x\, p(x)) \to q \vdash$ are not in Class 1 since they have a positive occurrence of $\to$ and $\forall$, resp.

Many practically relevant problems are in Class 1; all Horn problems, all rewriting problems, and all problems in CNF are in Class 1. It is instructive to look at the proof that intuitionistic provability is equivalent to classical provability for all problems in Class 1:

Theorem 4 ([30]). *Let $\Gamma \vdash \Delta$ by a sequent in Class 1. If $\Gamma \vdash \Delta$ is provable in classical logic, then it is provable in intuitionistic logic as well.*

Proof. Let π be a cut-free proof of $\Gamma \vdash \Delta$ in the sequent calculus LK (which is cut-free complete for classical logic). Then π does not contain any of the inferences $\to_r$ or $\forall_r$ by the subformula property (these are the only inferences that are different between LK and mLJ), and is hence a proof in mLJ. □

The following corollary now shows the completeness of the constructivization procedure and hence our prover as a whole for sequents in Class 1:

Corollary 2. *Let E be an expansion proof such that* $\mathrm{sh}(E)$ *is in Class 1, and* $\mathrm{dp}(E)$ *is a quasi-tautology. Then* Construct*(E) returns true.*

Proof. There is a weakly regular proof in LK realizing E since $\mathrm{dp}(E)$ is a quasi-tautology, with the observation in Theorem 4 this proof is in mLJ. Hence Construct succeeds by Theorem 3. □

A similar result also holds for Orevkov's Class 2 (no positive occurrences of $\to$ and no negative occurrences of $\vee$). The constructivization procedure of Gilbert [17] was also shown to be complete for Class 2 (called F in their paper).

7 Experimental evaluation

We have implemented and evaluated this constructivization approach in the open source GAPT[3] system for proof theory [16], version 2.14. Many of its features are centered around a computational implementation of Herbrand's theorem and expansion trees, such as lemma generation [15], inductive theorem proving [12,14], deskolemization, and proof import [35].

The intuitionistic first-order prover based on this constructivization procedure is called Slakje, and provides a command-line program reading input problems in TPTP format [37]. Since GAPT is written in Scala and distributed as a platform-neutral tarball, we want to avoid external dependencies and prefer to use libraries available on the JVM: as a SAT solver we use Sat4j [3], for equality reasoning we wrote a simple congruence closure implementation.

GAPT already contains a reliable interface to external theorem provers that produces expansion proofs, supporting many first-order provers, including Vampire [20], E [36], SPASS [42], leanCoP [32], Prover9 [23], as well as others. GAPT also includes a simple built-in superposition prover called Escargot, which is mainly used for proof replay and small-scale automation in tactic proofs. For the experimental evaluation, we used Vampire 4.2.2, E 2.2, and Escargot as backends for Slakje.

The prover interface in GAPT supports most external provers (including Vampire and E) using proof replay, which reconstructs the proofs line-by-line by reproving each inference as a first-order problem using Escargot. There is special support for the Avatar [41] splitting inferences produced by Vampire. The interface operates on the level of clauses, the clausification and Skolemization is performed inside GAPT. Parsing and importing the Skolemization and clausification steps of all supported provers would be a tremendous amount of work, since every prover (and sometimes different versions of the same prover) use different proof output for these steps.

We evaluated the Slakje prover on the problems in the first-order section of the Intuitionistic Logic Theorem Proving library [34]. The ILTP library contains

[3] Open source, and freely available at https://logic.at/gapt

a mixture of problems from the TPTP, as well as problems designed for intuitionistic provers in the `GEJ` (constructive geometry), `GPJ` (group theory), and `SYJ` (intuitionistic syntactic) categories. The ILTP also contains benchmarking results for a number of intuitionistic theorem provers. Later on, the provers imogen [24] and WhaleProver [33] were also benchmarked on the ILTP.

Unfortunately, we were unable to compile and run many of these provers on a modern Linux distribution. Some depend on an old version of ECLIPSe prolog, which is no longer maintained and also no longer compiles. Others are written for Standard ML, but they do not compile with the current MLton version. For this reason, we used the publicly available timing information distributed alongside the ILTP[4] and the imogen source code[5]. To account for the difference in hardware, we ran Slakje with a smaller time limit of one minute (the ILTP benchmarks had a time limit of 10 minutes). A similar approach was already used in the evaluation of WhaleProver [33] (we do not evaluate WhaleProver here as neither the implementation nor the results are publicly available; according to the statistics in the paper its performance is comparable to imogen).

We conducted the benchmarks on a Debian Linux system with an Intel i5-4570 CPU and 8 GiB RAM. For comparison, the ILTP benchmarks were run on a Intel Xeon CPU with 3.4 GHz and the Mandrake 10.2 distribution.

The ILTP contains 2670 problems in the first-order section. Slakje solves 1674 of these problems (1290 theorems and 384 non-theorems) with Vampire[6] as a backend (total time limit of 60 seconds). The next best provers are ileanCoP and imogen, ileanCoP solves 690 (610 theorems and 80 non-theorems), imogen solves 865 (646 theorems and 219 non-theorems). 831 of the problems solved by Slakje were not solved by any other prover in the ILTP or imogen. Slakje could not solve 115 problems that were solved by another prover in the ILTP or imogen, 85 of these problems are intuitionistic non-theorems, only 30 problems were missed because we could not constructivize the classical proof.

The runtime of runtime of Slakje with the three backends (Escargot, E, and Vampire), the provers benchmarked in the ILTP, as well as imogen, is shown as a cactus plot in Fig. 2. Slakje is leading with any of the three backends; the most problems are solved using Vampire (1674), followed by E (1580), and Escargot (1459).

We might assume that the success of Slakje is due to benchmark set: that the ILTP contains many Horn problems or purely equational problems. However this is not the case. Only 650 of the problems in the ILTP are in Class 1 (recall Definition 8). We also looked at the formulas that were actually used in the classical proofs: considering only the actually used formulas, 980 of the problems are in Class 1. Looking at the runtime plots for Class 1 problems vs. non-Class 1 problems, there does not seem to be a large difference and Slakje is also leading even for the harder non-Class 1 problems.

[4] in the `ILTP-v1.1.2-fof-comparison.txt` file
[5] in the `IltpFol-1289.reg` file
[6] Version 4.2.2 from https://vprover.github.io/, no command-line options.

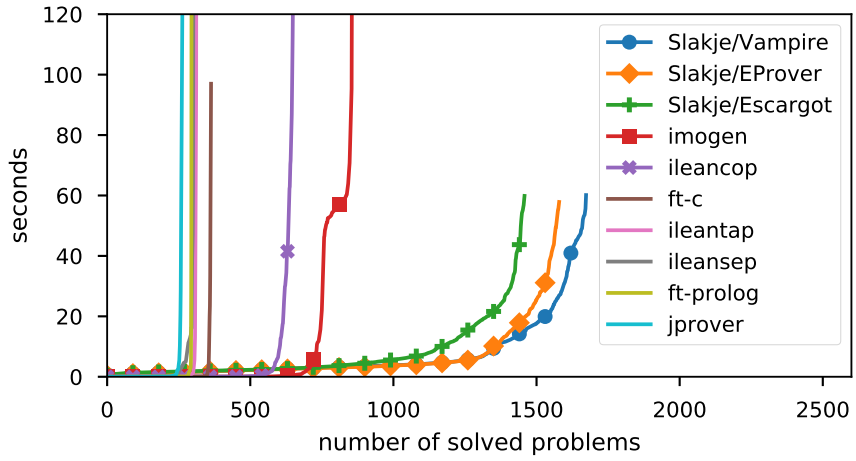

Fig. 2. Cactus plot of the runtime on solved problems.

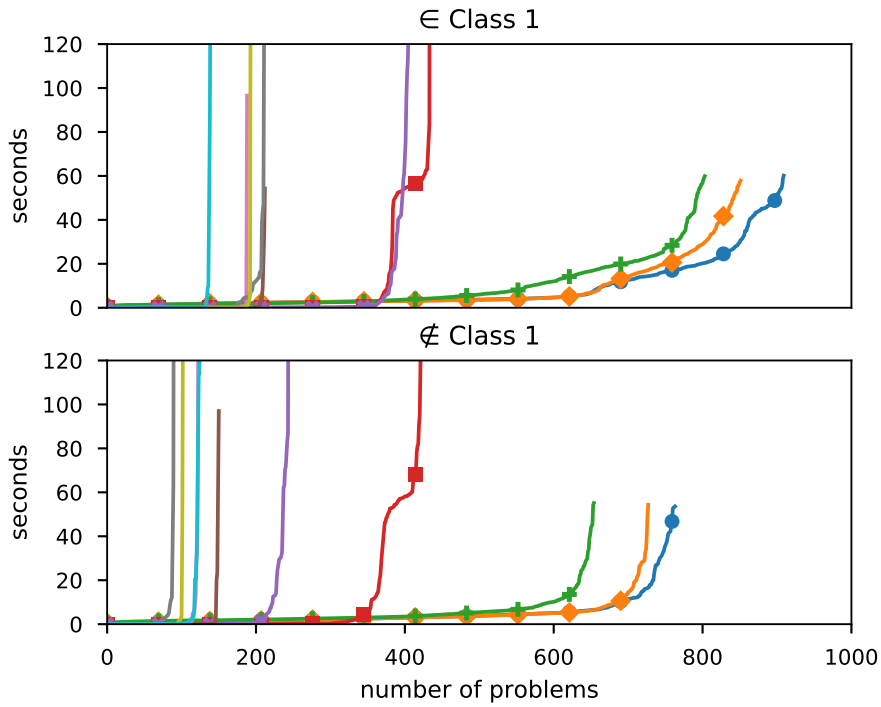

Fig. 3. Runtime comparison on problems in Class 1 vs. problems not in Class 1. (For problems where we could import a proof from the classical first-order prover, we only considered the formulas that were actually used to determine whether the problem is in Class 1.)

We have also run Vampire directly on the problems in the ILTP (in CASC mode with a time limit of 60 seconds) to obtain a realistic upper limit on how many problems we can expect to solve intuitionistically. In this configuration Vampire solves 2468 problems (2079 proofs and 389 satisfiable). When used via GAPT's prover interface, Vampire solves 2184 problems, returning 1830 proofs in the textual TPTP derivation format and 354 satisfiable results. Proof replay then produces 1786 resolution proofs, which are converted to 1771 expansion proofs, ultimately yielding 1290 intuitionistic proofs in mLJ. In each step we lose a few proofs due to the time limit. The largest difference is in the initial step of running the external theorem prover. We believe that this is mainly due to two reasons: first, the TPTP parser in GAPT is less efficient and takes a long time to parse larger problems. Second, we run Vampire in the default mode instead of the CASC mode, since the CASC mode produces proofs that we cannot parse reliably, making it less effective than the default mode in our interface.

8 Conclusion

First-order theorem proving seems to be fundamentally easier in classical logic than in intuitionistic logic. We can use Skolemization, and have CNFs as a simple normal form. The practical proof constructivization procedure that we have presented allows to reuse some of these advantages of classical logic. In a sense, we are learning from classical proofs to produce intuitionistic ones. In our setting, we are learning the quantifier instances. On an empirical level, we have shown that these instances as captured by expansion trees provide enough information to produce intuitionistic proofs.

This proof constructivization technique is so effective that we obtain a highly competitive automated intuitionistic first-order theorem prover by combining it with a classical theorem prover. This prover, Slakje, performs very well on the ILTP benchmark library for intuitionistic theorem provers: it solves 1674 out of 2670 problems (1290 theorems and 384 non-theorems) with Vampire as a backend, including 831 previously unsolved problems. This is almost twice as many solved problems than other state-of-the-art provers such as ileanCoP (solving 690 problems) and imogen (solving 865 problems).

However, this intuitionistic prover is incomplete since the classical theorem prover may not produce enough quantifier instances for an intuitionistic proof. One idea to fix this incompleteness that was already suggested by [10] is to add a complete instantiation strategy akin to the support for first-order reasoning in SMT solvers. Another approach would be to investigate variants of sound (semantic) translations of intuitionistic logic into classical logic which are optimized for automated theorem provers, and constructivize proofs of these translations.

As future work, we intend to integrate this prover approach in interactive theorem proving systems and evaluate its use for proof automation and as a strong reconstruction tactic for hammers in proof assistants based on intuitionistic logic.

References

1. Baaz, M., Ebner, G., Hetzl, S., Lolic, A., Weller, D.: Skolemization and equality (2019), in preparation
2. Baaz, M., Hetzl, S., Weller, D.: On the complexity of proof deskolemization. Journal of Symbolic Logic **77**(2), 669–686 (2012)
3. Berre, D.L., Parrain, A.: The Sat4j library, release 2.2. JSAT **7**(2-3), 59–6 (2010)
4. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development – Coq'Art: The Calculus of Inductive Constructions. Texts in Theoretical Computer Science, Springer (2004)
5. Bibel, W.: Matings in matrices. Communications of the ACM **26**(11), 844–852 (1983)
6. Blanchette, J.C., Bulwahn, L., Nipkow, T.: Automatic proof and disproof in Isabelle/HOL. In: Tinelli, C., Sofronie-Stokkermans, V. (eds.) Frontiers of Combining Systems, 8th International Symposium, FroCoS. Lecture Notes in Computer Science, vol. 6989, pp. 12–27. Springer (2011)
7. Bove, A., Dybjer, P., Norell, U.: A brief overview of Agda - A functional language with dependent types. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOL. Lecture Notes in Computer Science, vol. 5674, pp. 73–78. Springer (2009)
8. Buss, S.R.: On Herbrand's Theorem. In: Logic and Computational Complexity, Lecture Notes in Computer Science, vol. 960, pp. 195–209. Springer (1995)
9. Cauderlier, R.: A rewrite system for proof constructivization. In: Dowek, G., Licata, D.R., Alves, S. (eds.) 11th Workshop on Logical Frameworks and Meta-Languages: Theory and Practice, LFMTP. pp. 2:1–2:7. ACM (2016)
10. Claessen, K., Rosén, D.: SAT modulo intuitionistic implications. In: Davis, M., Fehnker, A., McIver, A., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning, 20th International Conference, LPAR-20. Lecture Notes in Computer Science, vol. 9450, pp. 622–637. Springer (2015)
11. Czajka, L., Kaliszyk, C.: Hammer for Coq: Automation for dependent type theory. Journal of Automated Reasoning **61**(1-4), 423–453 (2018)
12. Eberhard, S., Hetzl, S.: Inductive theorem proving based on tree grammars. Annals of Pure and Applied Logic **166**(6), 665–700 (2015)
13. Ebner, G.: Extracting expansion trees from resolution proofs with splitting and definitions (2018), preprint available at https://gebner.org/pdfs/2018-01-29_etimport.pdf
14. Ebner, G., Hetzl, S.: Induction formulas and term structure (2019), submitted
15. Ebner, G., Hetzl, S., Leitsch, A., Reis, G., Weller, D.: On the generation of quantified lemmas. Journal of Automated Reasoning pp. 1–32 (2018)
16. Ebner, G., Hetzl, S., Reis, G., Riener, M., Wolfsteiner, S., Zivota, S.: System description: GAPT 2.0. In: Olivetti, N., Tiwari, A. (eds.) International Joint Conference on Automated Reasoning (IJCAR). Lecture Notes in Computer Science, vol. 9706, pp. 293–301. Springer (2016)
17. Gilbert, F.: Automated constructivization of proofs. In: Esparza, J., Murawski, A.S. (eds.) Foundations of Software Science and Computation Structures, FOSSACS. Lecture Notes in Computer Science, vol. 10203, pp. 480–495 (2017)
18. Herbrand, J.: Recherches sur la théorie de la démonstration. Ph.D. thesis, Université de Paris (1930)
19. Kaliszyk, C., Urban, J.: Learning-assisted automated reasoning with Flyspeck. Journal of Automated Reasoning **53**(2), 173–213 (2014)

20. Kovács, L., Voronkov, A.: First-order theorem proving and Vampire. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification, 25th International Conference, CAV. Lecture Notes in Computer Science, vol. 8044, pp. 1–35. Springer (2013)
21. Liang, C., Miller, D.: Focusing and polarization in linear, intuitionistic, and classical logics. Theoretical Computer Science **410**(46), 4747–4768 (2009)
22. Maehara, S.: Eine Darstellung der intuitionistischen Logik in der Klassischen. Nagoya Mathematical Journal **7**, 45–64 (1954)
23. McCune, W.: Prover9 and Mace4 (2005–2010), http://www.cs.unm.edu/~mccune/prover9/
24. McLaughlin, S., Pfenning, F.: Efficient intuitionistic theorem proving with the polarized inverse method. In: Schmidt, R.A. (ed.) 22nd International Conference on Automated Deduction, CADE. Lecture Notes in Computer Science, vol. 5663, pp. 230–244. Springer (2009)
25. Miller, D.A.: A compact representation of proofs. Studia Logica **46**(4), 347–370 (1987)
26. Mints, G.: Gentzen-type system and resolution rules. Part I: Propositional logic. In: Martin-Löf, P., Mints, G. (eds.) COLOG 1988. vol. 417, pp. 198–231 (1988)
27. Mints, G.: Gentzen-type system and resolution rules. Part II: Propositional logic. In: Logic Colloquium 1990 (1993)
28. de Moura, L.M., Kong, S., Avigad, J., van Doorn, F., von Raumer, J.: The Lean theorem prover (system description). In: Felty, A.P., Middeldorp, A. (eds.) 25th International Conference on Automated Deduction, CADE. Lecture Notes in Computer Science, vol. 9195, pp. 378–388. Springer (2015)
29. Negri, S.: Glivenko sequent classes in the light of structural proof theory. Archive for Mathematical Logic **55**(3-4), 461–473 (2016)
30. Orevkov, V.P.: On Glivenko sequent classes. Trudy Matematicheskogo Instituta imeni V. A. Steklova **98**, 131–154 (1968)
31. Otten, J.: Clausal connection-based theorem proving in intuitionistic first-order logic. In: Beckert, B. (ed.) Automated Reasoning with Analytic Tableaux and Related Methods, TABLEAUX. Lecture Notes in Computer Science, vol. 3702, pp. 245–261. Springer (2005)
32. Otten, J.: leanCoP 2.0 and ileanCoP 1.2: High performance lean theorem proving in classical and intuitionistic logic (system descriptions). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) Automated Reasoning, 4th International Joint Conference, IJCAR 2008. Lecture Notes in Computer Science, vol. 5195, pp. 283–291. Springer (2008)
33. Pavlov, V., Pak, V.: WhaleProver: First-order intuitionistic theorem prover based on the inverse method. In: Petrenko, A.K., Voronkov, A. (eds.) Perspectives of System Informatics - 11th International Andrei P. Ershov Informatics Conference, PSI. Lecture Notes in Computer Science, vol. 10742, pp. 322–336. Springer (2017)
34. Raths, T., Otten, J., Kreitz, C.: The ILTP problem library for intuitionistic logic. Journal of Automated Reasoning **38**(1-3), 261–271 (2007)
35. Reis, G.: Importing SMT and connection proofs as expansion trees. In: Fourth Workshop on Proof eXchange for Theorem Proving, PxTP. pp. 3–10 (2015)
36. Schulz, S.: System description: E 1.8. In: McMillan, K.L., Middeldorp, A., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning, 19th International Conference, LPAR-19. Lecture Notes in Computer Science, vol. 8312, pp. 735–743. Springer (2013)
37. Sutcliffe, G.: The TPTP problem library and associated infrastructure: The FOF and CNF parts, v3.5.0. Journal of Automated Reasoning **43**(4), 337–362 (2009)

38. Troelstra, A.S., Schwichtenberg, H.: Basic Proof Theory. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press (2000)
39. Univalent Foundations Program, T.: Homotopy Type Theory: Univalent Foundations of Mathematics. Institute for Advanced Study (2013), https://homotopytypetheory.org/book
40. Voronkov, A.: Proof-search in intuitionistic logic with equality, or back to simultaneous rigid E-unification. Journal of Automated Reasoning **30**(2), 121–151 (2003)
41. Voronkov, A.: AVATAR: the architecture for first-order theorem provers. In: Biere, A., Bloem, R. (eds.) 26$^{\text{th}}$ International Conference on Computer Aided Verification, CAV 2014. Lecture Notes in Computer Science, vol. 8559, pp. 696–710. Springer (2014)
42. Weidenbach, C., Dimova, D., Fietzke, A., Kumar, R., Suda, M., Wischnewski, P.: SPASS version 3.5. In: Schmidt, R.A. (ed.) 22nd International Conference on Automated Deduction, CADE-22. Lecture Notes in Computer Science, vol. 5663, pp. 140–145. Springer (2009)