Herbrand Constructivization for Automated Intuitionistic Theorem Proving

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 800 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 [3], Agda [6], or Lean [26]. In some foundational frameworks the law of excluded middle is even provably false, such as in homotopy type theory[1] [36]. 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 [29]. 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 [8] and Gilbert [16]; 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 [5,10,18]. In this framework, proof constructivization also uses an unsound translation: one that maps each formula to itself (but reinterprets it in a different logic).

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

We present a new and different proof constructivization method based on Herbrand disjunctions. Herbrand's theorem [17,7] 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. We say that t_i is a quantifier instance term for $\exists x\, \varphi(x)$. 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 [23].

- 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 [15] 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.

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 $\to, \wedge, \vee, \bot, \top$ and the quantifiers $\exists, \forall$. The abbreviation $\neg\varphi$ stands for $\varphi \to \bot$. 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 classical prover (e.g. Vampire [19]) with φ[2]
   (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; in our implementation, phases 1 and 2 are part of the general-purpose external prover interface that produces expansion proofs available in the GAPT [15,13] framework. We use expansion proofs as a compact intermediate format for classical

[2] Internally, Vampire, and in general most classical provers then refute $\neg\varphi$.

proofs, which only contain the quantifier instance terms but not the propositional reasoning in the proof. Many automated theorem proving paradigms generate proofs that directly 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 also 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 [24,25], 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* [1] to eliminate Skolemization from classical proofs (which is a natural operation on expansion proofs).

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 a propositional formula φ in classical logic, only the part where we need to prove $\varphi \to \bigwedge C$ for the CNF $\bigwedge C$ requires classical logic. If the CNF $\bigwedge C$ is unsatisfiable, then $\bigwedge C$ is already provable in intuitionistic logic: proofs produced by SAT solvers can be translated to resolution proofs; and resolution is just the cut inference, which is sound for intuitionistic logic. (See also Theorem 4 for a different explanation.) In our setting, a SAT solver hence decides the following question: "is the sequent $\Gamma \vdash \Delta$ 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 [9] 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 [23] 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, storing 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.

Expansion trees have two polarities, $-$ and $+$. We write $-p$ for the inverse polarity of p, i.e. $-- = +$ and $-+ = -$. Polarity only changes on the left side of the connective $\rightarrow$. This distinction is important since we must instantiate positive occurrences of $\forall$ (resp. negative occurrences of $\exists$, called "strong quantifiers") with an eigenvariable, while we can instantiate the negative ones with whatever terms we want ("weak quantifiers"). An atom is a predicate such as $P(x, y)$ or an equality; the formulas $\top, \bot$ are not atoms.

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

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

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

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

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

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

Example 1. Consider the formula[3] $\varphi := \forall x\, P(x) \rightarrow (\forall x\, P(f(x)) \rightarrow Q) \rightarrow Q$. The expansion tree $\mathrm{ET}^+(\varphi) \ni E := (\forall x\, P(x) +^{f(\alpha)} P(f(\alpha))) \rightarrow (\forall x\, P(f(x)) +^{\alpha}_{\mathrm{ev}} P(f(\alpha)) \rightarrow Q) \rightarrow Q$ has the shallow formula $\mathrm{sh}(E) = \varphi$, and its deep formula $\mathrm{dp}(E) = (P(f(\alpha)) \rightarrow (P(f(\alpha)) \rightarrow Q) \rightarrow Q)$ 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.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P(f(\alpha)) \vdash P(f(\alpha))}{\forall x\, P(x) \vdash P(f(\alpha))}\,\forall_l}{\forall x\, P(x) \vdash \forall x\, P(f(x))}\,\forall_r \qquad Q \vdash Q}{\forall x\, P(x), \forall x\, P(f(x)) \rightarrow Q \vdash Q}\,\rightarrow_l}{\forall x\, P(x) \vdash (\forall x\, P(f(x)) \rightarrow Q) \rightarrow Q}\,\rightarrow_r}{\vdash \forall x\, P(x) \rightarrow (\forall x\, P(f(x)) \rightarrow Q) \rightarrow Q}\,\rightarrow_r$$

[3] We use the convention that the quantifiers $\forall, \exists$ bind stronger than $\rightarrow, \wedge, \vee$. That is, $\forall x\, P(x) \rightarrow Q$ is the same formula as $(\forall x\, P(x)) \rightarrow Q$. Furthermore, $\rightarrow$ is right-associative, that is, $P \rightarrow Q \rightarrow R$ is the same formula as $P \rightarrow (Q \rightarrow R)$.

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

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

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

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

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

The deep formula corresponds to the Herbrand disjunction. In an expansion proof, the eigenvariables need to be acyclic. This restriction is similar to the eigenvariable condition in sequent calculi and the acceptability condition for substitutions in matrices [4]. Let $\mathrm{FV}(\varphi)$ be the set of free variables of a formula φ.

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

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

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

Theorem 1 ([23, Theorems 4.1 and 4.2]). *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 $\exists x\, (p(c) \lor p(d) \to p(c))$ has $E_1 = \exists x\, (p(c) \lor p(d) \to p(c)) +^c (p(c)^- \lor p(d)^- \to p(c)^+)$ as an expansion proof. The deep formula $\mathrm{dp}(E_1) = p(c) \lor p(d) \to p(c)$ is a tautology. This is not the only possible expansion proof of this formula: we could also use the instance d instead of c.

Expansion proofs are closely related to the matrix characterization for classical first-order logic [4] used by connection-based theorem provers [30]. 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. In the classical setting, the multiplicity in a connection proof corresponds essentially to the number of children in the weak quantifier nodes of an expansion tree. (In the intuitionistic setting, the multiplicity also constrains the amount of contraction in a corresponding sequent calculus proof, i.e., how often subformulas can be used. There is no such constraint in our expansion-proof based method.) Integrating equality into connection proofs is hard as it requires simultaneous rigid E-unification [38], and

[4] Proof systems (for propositional logic) are typically required to be polynomial-time checkable, as certificates to the coNP-complete validity problem. Expansion proofs are coNP-checkable certificates for the undecidable first-order validity problem.

connection provers such as leanCoP hence perform equational reasoning during proof search by adding axioms for reflexivity, transitivity, and congruence of equality during preprocessing.

By contrast, expansion proofs work modulo equality. We do not need to 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 (which are typically the fastest tools to decide validity of quantifier-free formulas). 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. The expansion proof only restricts the quantifier instance *terms* in the proof; not how often a subformula is used (i.e. contraction). We want to find a proof in the multi-succedent intuitionistic sequent calculus mLJ as shown in Fig. 1, where all eigenvariables and quantifier instances occur in the expansion proof and there are no duplicate eigenvariables along any branch of the proof.

Definition 5. *An mLJ-proof π* realizes *an expansion proof E iff every quantifier instance term in π is contained in E, i.e.: (and analogously for $\exists$)*

- *If $\dfrac{\Gamma \vdash \varphi(\alpha)}{\Gamma \vdash \forall x\, \varphi(x)}$ is a subproof of π, then $\forall x\, \varphi(x) +^{\alpha}_{\mathrm{ev}} E'$ 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')*

Note that Definition 5 ignores the ancestor relationship of formulas in a proof: if two subtrees E_1, E_2 of E have the same shallow formula, then their instances can be be used interchangeably in π.

Definition 6. *An mLJ-proof π is called* weakly regular *iff for all subproofs of the following form, α does not occur as the eigenvariable of an inference in π':*

$$\dfrac{\overset{(\pi')}{\Gamma \vdash \varphi(\alpha)}}{\Gamma \vdash \forall x\, \varphi(x)}\,\forall_r \qquad \textit{or} \qquad \dfrac{\overset{(\pi')}{\varphi(\alpha), \Gamma \vdash \Delta}}{\exists x\, \varphi(x), \Gamma \vdash \Delta}\,\exists_l$$

Our algorithm will have the property that whenever a cut-free weakly regular mLJ-proof of $\mathrm{sh}(E)$ 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.

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

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

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

$$\frac{}{\vdash \top}\,\top_r \qquad \frac{}{\bot \vdash}\,\bot_l \qquad \frac{\Gamma \vdash \Delta, \varphi, \psi}{\Gamma \vdash \Delta, \varphi \lor \psi}\,\lor_r \qquad \frac{\varphi, \Gamma \vdash \Delta \qquad \psi, \Gamma \vdash \Delta}{\varphi \lor \psi, \Gamma \vdash \Delta}\,\lor_l$$

$$\frac{\varphi, \psi, \Gamma \vdash \Delta}{\varphi \land \psi, \Gamma \vdash \Delta}\,\land_l \qquad \frac{\Gamma \vdash \Delta, \varphi \qquad \Gamma \vdash \Delta, \psi}{\Gamma \vdash \Delta, \varphi \land \psi}\,\land_r$$

$$\frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \rightarrow \psi}\,\rightarrow_r \qquad \frac{\Gamma \vdash \Delta, \varphi \qquad \psi, \Gamma \vdash \Delta}{\varphi \rightarrow \psi, \Gamma \vdash \Delta}\,\rightarrow_l$$

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

Fig. 1. The multi-succedent calculus mLJ for intuitionistic first-order logic (variant of L'J first introduced by Maehara [20] but using sets instead of sequences, see also mG1i in Troelstra and Schwichtenberg's classification [37]). A sequent $\Gamma \vdash \Delta$ consists of two sets of formulas Γ and Δ and is interpreted as the formula $\bigwedge \Gamma \rightarrow \bigvee \Delta$. The variable α in the $\exists_l$ and $\forall_r$ inferences is called an eigenvariable, and must not occur in Γ, Δ as a free variable. The proof system is cut-free complete for intuitionistic first-order logic with equality. Note that the rules $\rightarrow_r, \forall_r$ do not have any extra formulas Δ in the succedent: this is the only difference to the classical calculus.

Example 3. Consider the expansion proof $\vdash \neg\neg(\forall x\ (p \lor \neg p) +^{\alpha}_{\mathrm{ev}} (p^+ \lor \neg p^-))$. This expansion proof cannot be realized by a weakly regular cut-free mLJ-proof of $\vdash \neg\neg\forall x\ (p \lor \neg p)$, 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. (Note that this example requires that α does not occur in p: the formula $\neg\neg\forall x\ (q(x) \lor \neg q(x))$ is not an intuitionistic theorem.)

The SAT-based bottom-up 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 $\rightarrow_r$ 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 by obtaining a model from the SAT solver. Such a sequent corresponds to a leaf in a restricted bottom-up search, which only uses inferences (all except for $\rightarrow_r, \forall_r, \exists_l$—these inferences have an eigenvariable or single-conclusion restriction) that we have encoded as clauses in the SAT solver (in the Construct function). There may be multiple such leaves (e.g. corresponding to two $\forall_r$ inferences in different branches of the proof), hence Solve iterates over the models in a loop.

We use an incremental interface to the SAT solver: the solver internally maintains a set of clauses $\mathcal{C}$. A clause is a set of literals. If l is a literal, then $-l$ is the negated literal. The function Assert adds a clause to this set. Given a

Algorithm 1 SAT-based proof constructivization

```
procedure CONSTRUCT($E$)    ▷ returns true if intuitionistic proof of sh($E$) found
    ASSERT($\|\top\|$);  ASSERT($-\|\bot\|$)
    ASSERT($-\|\mathrm{sh}(E_1)\|, \|\mathrm{sh}(E_1 \lor E_2)\|$) for each subtree $E_1 \lor E_2$ of $E$
    ASSERT($-\|\mathrm{sh}(E_2)\|, \|\mathrm{sh}(E_1 \lor E_2)\|$) for each subtree $E_1 \lor E_2$ of $E$
    ASSERT($-\|\mathrm{sh}(E_1 \lor E_2)\|, \|\mathrm{sh}(E_1)\|, \|\mathrm{sh}(E_2)\|$) for each subtree $E_1 \lor E_2$ of $E$
    ASSERT($-\|\mathrm{sh}(E_1 \land E_2)\|, \|\mathrm{sh}(E_1)\|$) for each subtree $E_1 \land E_2$ of $E$
    ASSERT($-\|\mathrm{sh}(E_1 \land E_2)\|, \|\mathrm{sh}(E_2)\|$) for each subtree $E_1 \land E_2$ of $E$
    ASSERT($-\|\mathrm{sh}(E_1)\|, -\|\mathrm{sh}(E_2)\|, \|\mathrm{sh}(E_1 \land E_2)\|$) for each subtree $E_1 \land E_2$ of $E$
    ASSERT($-\|\mathrm{sh}(E_2)\|, \|\mathrm{sh}(E_1 \to E_2)\|$) for each subtree $E_1 \to E_2$ of $E$
    ASSERT($-\|\mathrm{sh}(E_1)\|, -\|\mathrm{sh}(E_1 \to E_2)\|, \|\mathrm{sh}(E_2)\|$) for each subtree
                                                             $E_1 \to E_2$ of $E$
    ASSERT($-\|\mathrm{sh}(E')|, \|\mathrm{sh}(E_i)\|$) for each subtree
                             $E' = \forall x\, \varphi(x) +^{t_1} E_1 \cdots +^{t_n} E_n$ of $E$ and $1 \le i \le n$
    ASSERT($-\|\mathrm{sh}(E_i)\|, \|\mathrm{sh}(E')\|$) for each subtree
                             $E' = \exists x\, \varphi(x) +^{t_1} E_1 \cdots +^{t_n} E_n$ of $E$ and $1 \le i \le n$
    return SOLVE($E; \emptyset; \vdash \mathrm{sh}(E)$)
procedure SOLVE($E; \Sigma; \Gamma \vdash \Delta$)    ▷ $\Sigma$ is the set of already used eigenvariables
                    ▷ returns true if we have found an intuitionistic proof of $\Gamma \vdash \Delta$
    while ISESATISFIABLE($\|\Gamma\|, -\|\Delta\|$) do
        $I$ = GETMODEL($\|\Gamma\|, -\|\Delta\|$)
        $(\Gamma', \Delta') = (\{\varphi \mid I(\|\varphi\|) = 1\}, \{\varphi \mid I(\|\varphi\|) = 0\})$    ▷ $\Gamma' \supseteq \Gamma$ and $\Delta' \supseteq \Delta$
        for each $\varphi \to \psi$ in $\Delta'$ such that $\varphi \notin \Gamma'$ do
            if SOLVE($E; \Sigma; \Gamma', \varphi \vdash \psi$) then
                $C$ = UNSATCORE($\|\Gamma'\|, \|\varphi\|, -\|\psi\|$)
                ASSERT($-C \setminus \{\|\psi\|, -\|\varphi\|\}, \|\varphi \to \psi\|$)
                continue outer loop
        for each $\forall x\, \varphi$ in $\Delta'$ and subtree $\forall x\, \varphi +^{\alpha}_{\mathrm{ev}} \ldots$ in $E$ with $\alpha \notin \Sigma$ do
            $\Gamma'_\alpha = \{\psi \in \Gamma' \mid \alpha \notin \mathrm{FV}(\psi)\}$
            if SOLVE($E; \Sigma \cup \{\alpha\}; \Gamma'_\alpha \vdash \varphi(\alpha)$) then
                $C$ = UNSATCORE($\|\Gamma'_\alpha\|, -\|\varphi(\alpha)\|$)
                ASSERT($-C \setminus \{\|\varphi(\alpha)\|\}, \|\forall x\, \varphi(x)\|$)
                continue outer loop
        for each $\exists x\, \varphi$ in $\Gamma'$ and subtree $\exists x\, \varphi +^{\alpha}_{\mathrm{ev}} \ldots$ in $E$ with $\alpha \notin \Sigma$ do
            $(\Gamma'_\alpha, \Delta'_\alpha) = (\{\psi \in \Gamma' \mid \alpha \notin \mathrm{FV}(\psi)\}, \{\psi \in \Delta' \mid \alpha \notin \mathrm{FV}(\psi)\})$
            if SOLVE($E; \Sigma \cup \{\alpha\}; \Gamma'_\alpha, \varphi(\alpha) \vdash \Delta'_\alpha$) then
                $C$ = UNSATCORE($\|\Gamma'_\alpha\|, \|\varphi(\alpha)\|, -\|\Delta'_\alpha\|$)
                ASSERT($-C \setminus \{-\|\varphi(\alpha)\|\}, -\|\exists x\, \varphi(x)\|$)
                continue outer loop
        return false
    return true
procedure ISESATISFIABLE($A$)    ▷ returns true iff satisfiable modulo equality
    while ISSATISFIABLE($A$) do    ▷ implemented using congrence closure
        $I$ = GETMODEL($A$)
        if $\{\varphi \mid I(\|\varphi\|) = 1\} \vdash \{\varphi \mid I(\|\varphi\|) = 0\}$ is provable with cut, w, rfl, eq then
            ASSERT(end-sequent of equality proof)
        else return true
    return false
```

set of literals A (short for assumptions), the function IsSatisfiable(A) returns true iff $\bigwedge \mathcal{C} \wedge \bigwedge A$ is satisfiable. If it is satisfiable, the function GetModel(A) returns a model. If it is unsatisfiable, then UnsatCore(A) returns a minimal subset $A' \subseteq A$ such that $\bigwedge \mathcal{C} \wedge \bigwedge A'$ is still unsatisfiable.

Concretely we associate to the shallow formula φ of every subtree of E a variable $\|\varphi\|$ in the SAT solver. Given a set of formulas Γ, we also define $\|\Gamma\| = \{\|\varphi\| \mid \varphi \in \Gamma\}$ (in particular $\|\emptyset\| = \emptyset$). We only call Assert($-\|\Gamma\|, \|\Delta\|$) if we have an intuitionistic mLJ-proof of $\Gamma \vdash \Delta$ (that is, $\bigwedge \Gamma \to \bigvee \Delta$ is an intuitionistic theorem). A model I returned by the SAT solver corresponds to the sequent $\{\varphi \mid I(\|\varphi\|) = 1\} \vdash \{\varphi \mid I(\|\varphi\|) = 0\}$.

By asserting specific clauses, we can ensure that these sequents obtained from models are closed under inferences: for example if we call Assert($-\|\varphi \wedge \psi\|, \|\psi\|$) then any model I satisfies $I(\|\psi\|) = 1$ if $I(\|\varphi \wedge \psi\|) = 1$. Hence, the sequent $\Gamma \vdash \Delta$ corresponding to the model has $\psi \in \Gamma$ if $\varphi \wedge \psi \in \Gamma$ and is closed under (part of) the $\wedge_l$ rule (read bottom-up). We add these clauses in the Construct function.

The other inferences, $\to_r, \forall_r, \exists_l$, are handled in the Solve function. For example, lines 23-26 handle an $\to_r$ inference inferring $\Gamma' \vdash \varphi \to \psi$ from $\Gamma', \varphi \vdash \psi$ in the following way: the recursive Solve-call first tries to prove $\Gamma', \varphi \vdash \psi$. The set of literals C returned by UnsatCore then corresponds to a minimal subset $\Gamma'' \subseteq \Gamma'$ such that $\Gamma'', \varphi \vdash \psi$ (the minimization is purely an optimization, albeit an important one). The correspondence is that $\|\Gamma''\| = C \setminus \{\|\varphi\|, -\|\psi\|\}$. We then assert $-\|\Gamma''\|, \|\varphi \to \psi\|$ (computed using C in line 25) corresponding to the provable sequent $\Gamma'' \vdash \varphi \to \psi$. Note that the polarities of the SAT solver variables are inverse in the *assumptions* passed to UnsatCore and the *clause* passed to Assert: UnsatCore(A) returns a minimal subset of $A' \subseteq A$ such that $-A'$ (regarded as a clause) is implied by the current clauses.

Algorithm 1 terminates, since each recursive call of Solve either increases the size of Σ or increases the size of the antecedent Γ while keeping Σ the same; the while-loop in Solve never iterates over the same model twice since the Assert-calls in Solve add clauses that are not true in the current model.

Example 4. Consider again the expansion proof from Example 1 and abbreviate $\psi = \forall x\, P(f(x)) \to Q$. Then Construct asserts the following clauses:

$$\begin{array}{ll}
\|\top\| & -\|\varphi\|, -\|\forall x\, P(x)\|, \|\psi \to Q\| \\
-\|\bot\| & -\|\psi \to Q\|, -\|\psi\|, \|Q\| \\
-\|\psi \to Q\|, \|\varphi\| & -\|\psi\|, -\|\forall x\, P(f(x))\|, \|Q\| \\
-\|Q\|, \|\psi \to Q\| & -\|\forall x\, P(x)\|, \|P(f(\alpha))\|
\end{array}$$

And Solve proceeds in the following recursive call tree:

- Solve(E; $\emptyset$; $\vdash \varphi$)
  - Obtained model: $\|\top\|$ (list of all p with $I(p) = 1$)
  - Solve(E; $\emptyset$; $\top, \forall x\, P(x) \vdash \psi \to Q$) (line 23 for φ)
    - Obtained model: $\|\top\|, \|\forall x\, P(x)\|, \|P(f(\alpha))\|$
    - Solve(E; $\emptyset$; $\top, \forall x\, P(x), P(f(\alpha)), \psi \vdash Q$) (line 23 for $\psi \to Q$)

        · Obtained model: $\|\top\|, \|\forall x\, P(x)\|, \|P(f(\alpha))\|, \|\psi\|$
        · $\textsc{Solve}(E; \{\alpha\}; \top, \forall x P(x), \psi \vdash P(f(\alpha)))$ (line 29 for $\forall x P(f(x))$)
        · $\textsc{Assert}(-\|\forall x\, P(x)\|, \|\forall x\, P(f(x))\|)$
    * $\textsc{Assert}(-\|\forall x\, P(x)\|, \|\psi \to Q\|)$
- $\textsc{Assert}(\|\varphi\|)$

Theorem 2. *If* Construct*(E) returns true, then there is an mLJ-proof of* sh(E) *realizing E (and* sh(E) *is an intuitionistic theorem).*

Proof. We store an mLJ-proof $\Gamma \vdash \Delta$ for every clause $-\|\Gamma\|, \|\Delta\|$ that is passed to Assert: these all have straightforward proofs in mLJ. Whenever IsESatisfiable$(\|\Gamma\|, -\|\Delta\|)$ returns false for a sequent $\Gamma \vdash \Delta$, we have an mLJ-proof of $\Gamma \vdash \Delta$ by combining the previously stored proofs 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 of sh(E) 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 the 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 (for an expansion proof E) iff all of the following are true:*

- *$\Gamma \cup \Delta$ is the set of all shallow formulas of subtrees of 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 $\varphi(t_1), \dots, \varphi(t_n) \in \Gamma$.*
- *If $\exists x\, \varphi(x) \in \Delta$ and $\exists x\, \varphi(x) +_{t_1} \cdots +_{t_n}$ is a subtree of E, then $\varphi(t_1), \dots, \varphi(t_n) \in \Delta$*

Lemma 1. *The sequent $\Gamma' \vdash \Delta'$ obtained in line 21 from the model returned by* GetModel *in* Solve*$(E; \Sigma; \Gamma \vdash \Delta)$ is maximal for E.*

Proof. The set $\Gamma' \cup \Delta'$ contains all shallow formulas, and we have $\Gamma' \cap \Delta' = \emptyset$ because $\Gamma' \vdash \Delta'$ corresponds to a model. Each of the other conditions in Definition 7 is then ensured by a clause that is asserted in the Construct function: e.g. $\bot \in \Delta'$ is ensured by $\textsc{Assert}(-\|\bot\|)$. □

Lemma 2. *Let $\mathcal{S}$ be maximal for E, and π be an mLJ-proof of $\mathcal{S}'$ realizing E, such that $\mathcal{S}'$ 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:

$$\dfrac{\overset{(\pi_1)}{\Gamma \vdash \Delta, \varphi} \qquad \overset{(\pi_2)}{\Gamma \vdash \Delta, \psi}}{\Gamma \vdash \Delta, \varphi \wedge \psi} \wedge_r$$

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 mLJ-proof of* $\mathrm{sh}(E)$*, then* Construct*(E) returns true.*

Proof. We use the following invariant for Solve($E; \Sigma; \Gamma \vdash \Delta$): if there is a weakly regular cut-free mLJ-proof π of a subsequent of $\Gamma \vdash \Delta$ realizing E such that the eigenvariables in π are disjoint from Σ, then Solve returns true.

In Solve, let π be the subproof described above, and let $\Gamma' \vdash \Delta'$ be the sequent constructed from the model in line 21. Then $\Gamma' \vdash \Delta'$ is a maximal sequent by Lemma 1. There is a subproof π' of π whose end-sequent is a subsequent of $\Gamma' \vdash \Delta'$ and that ends in a $\exists_l, \forall_r$ or $\to_r$ inference by Lemma 2. At least one of the recursive calls then corresponds to this inference, and invokes Solve with the premise of the inference, which hence returns true. Weak regularity of π ensures that the precondition for Σ is satisfied. The clause passed to Assert corresponds to the conclusion of the $\exists_l, \forall_r, \to_r$-inference, and we continue to the next iteration of the loop. □

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 author believes that it should be possible to give a similar proof-theoretic completeness proof for the Intuit prover [9] (their paper gives a proof based on Kripke models).

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

5 Optimizations

For performance reasons, we implement several optimizations in the Solve procedure. The first one was already described in [9].

Caching unsolvable cases. By using a SAT solver, we already have a cache for the solvable cases: whenever Solve($E; \Sigma; \Gamma \vdash \Delta$) 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 line 21). At the beginning of Solve we check if we have already stored a pair $(\Sigma''; \Gamma'' \vdash \Delta'')$ such that $\Sigma \subseteq \Sigma''$ and $\Gamma \supseteq \Gamma''$ and $\Delta \supseteq \Delta''$, and return false if there is such a pair. (Pairs are stored in a trie-like data structure.)

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. The function IsESatisfiable($\|\Gamma\|, -\|\Delta\|$) returns false if there is a weakly regular cut-free mLJ-proof of $\Gamma \vdash \Delta$ realizing E *without* the inferences $\to_r, \forall_r, \exists_l$. We add a fresh variable c in such a way that IsESatisfiable($\mathsf{c}, \|\Gamma\|, -\|\Delta\|$) returns false iff there is a classical proof of $\Gamma \vdash \Delta$ realizing E. The classical calculus differs from mLJ only in the rules $\to_r, \forall_r$. (Concretely, we assert the clauses $\mathsf{c} \vdash \|\varphi \to \psi\|, \|\varphi\|$ and $\mathsf{c}, \|\varphi(\alpha)\| \vdash \|\forall x\, \varphi(x)\|$ and $\mathsf{c}, \|\exists x\, \varphi(x)\| \vdash \|\varphi(\alpha)\|$ for the corresponding shallow formulas of subtrees of E.)

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. Consider e.g. the expansion proof $\forall x\, \exists y\, R(x,y) +^{\alpha} (\cdots +^{\beta}_{\mathrm{ev}} R(\alpha,\beta)^-) +^{f(\alpha)} (\cdots +^{\gamma}_{\mathrm{ev}} R(f(\alpha),\gamma)^-) \vdash \forall x\ \exists y\ \exists z\ (R(x,y) \wedge R(f(x),z)) +^{\alpha}_{\mathrm{ev}} \cdots +^{\beta} \cdots +^{\gamma} (R(\alpha,\beta)^+ \wedge R(f(\alpha),\gamma)^+)$. In line 29, we first introduce the α eigenvariable, and then (in recursive calls) the β and γ eigenvariables in line 35. Without the optimization, we would then try *every permutation* of β, γ if Solve returned false (which will be expensive if there are more eigenvariables). With the optimization, we only need to consider a single permutation.

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 can use heuristic instantiation as a pre-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 [27] for a more modern presentation.

Definition 8 ([28]). Class 1 *is the set of sequents which do not have positive occurrences of $\to$ or $\forall$.*

Example 5. The sequent $\forall x\,(p(x) \to q(x) \lor \neg r(x)), p(c) \land r(c) \vdash q(c)$ 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 ([28]). *Let $\Gamma \vdash \Delta$ be 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. □

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 of $\mathrm{sh}(E)$ 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. □

Corollary 2 shows that our prover as a whole is complete for sequents in Class 1. A similar result also holds for Orevkov's Class 2 (no positive occurrences of $\to$ and no negative occurrences of $\lor$). The constructivization procedure of Gilbert [16] 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[5] system for proof theory [15], version 2.14. Many of its features are centered around a computational implementation of Herbrand's theorem and expansion trees, such as lemma generation [14], inductive theorem proving [12], deskolemization, and proof import [33].

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 [35]. 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 [2], for equality reasoning we wrote a simple congruence closure implementation.

By default, Slakje prints the generated mLJ proof in the TPTP derivation format, writing each sequent in the proof on a separate line. Other output options are supported as well: the `--prooftool` option displays the mLJ proof tree in

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

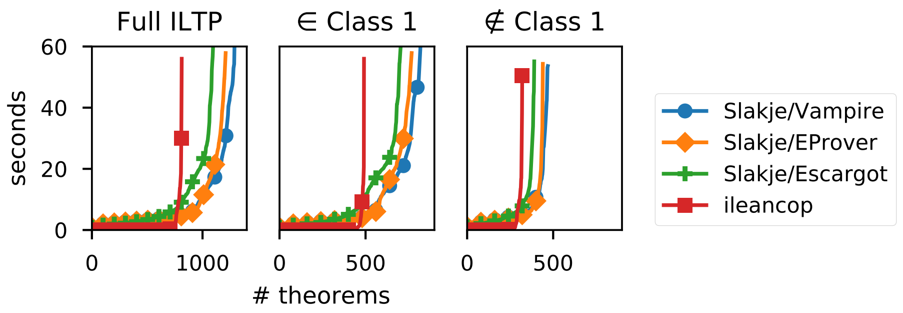

Fig. 2. Cactus plot of the prover runtime on proved theorems.

the graphical ProofTool user interface [11]. The `--lj` option transforms the mLJ proof to a cut-free proof in the single-conclusion intuitionistic LJ calculus.

GAPT already contains a reliable interface to external theorem provers that produces expansion proofs and supports many first-order provers, including Vampire [19], E [34], SPASS [40], leanCoP [30], Prover9 [21], 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. We do not call the external provers for quantifier-free problems: there are no quantifier instances that we could import, and furthermore the constructivization procedure is already a decision problem in the quantifier-free case by Corollary 1. 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 [39] 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 [32], version 1.1.2. The ILTP library contains 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 from 2006. In those results, ileanCoP [30] solves the largest number of problems by a significant margin.

The provers imogen [22] and WhaleProver [31] were also benchmarked on the ILTP, performing competitively with ileanCoP. According to [22], imogen solves 857 problems, improving on the 690 problems solved by ileanCoP 1.0. (On our hardware, the newest ileanCoP 1.2 version now solves 891 problems.)

Unfortunately we were not able to build a working version of imogen, so we could not benchmark it on our hardware. WhaleProver is not publicly available at all, according to [31] it solves 811 problems.

In our evaluation we compared Slakje against the current ileanCoP 1.2 version, running both Slakje and ileanCoP on a Debian Linux system with an Intel i5-4570 CPU and 8 GiB RAM.

The ILTP contains 2670 problems in the first-order section. (There are only 2550 problems according to the ILTP website, however the archive file[6] contains 2670 problem files.). Slakje solves 1674 of these problems (1290 theorems and 384 non-theorems) with Vampire 4.2.2 as a backend (total time limit of 60 seconds, no command-line options). ileanCoP 1.2 solves 891 (813 theorems and 78 non-theorems). 905 of the problems solved by Slakje were not solved by ileanCoP (546 of which are theorems, and 359 non-theorems). (Including the ILTP benchmark results as well, Slakje solves 800 problems not solved by ileanCoP in our benchmarks, or any prover in the 2006 ILTP benchmarks.) Slakje could not solve 122 problems that were solved by ileanCoP; 69 of these problems are intuitionistic non-theorems. For the other 53 intuitionistic theorems, in one case Slakje fails due to a timeout, and 24 could be solved with a different backend (Escargot or E).

The runtime of Slakje with the three backends (Escargot, E, and Vampire), and ileanCoP is shown as a cactus plot in Fig. 2. Slakje is leading in the number of proven theorems with any of the three backends; the most theorems are obtained using Vampire (1290 thms. and 384 nonthms.), followed by E (1210 thms. and 370 nonthms.), and Escargot (1096 thms. and 363 nonthms.).

While Slakje can prove many difficult problems, it has a high overhead: the median runtime for solved problems is 3199ms, compared to 46ms for ileanCoP. Within a time limit of one second, Slakje can only prove a single theorem, while ileanCoP proves 734. This overhead is likely due to multiple factors: since Slakje runs on the JVM, it takes some time for the just-in-time compiler to compile the code. Furthermore, the interface to the external provers such as Vampire was designed to be generic and is not highly optimized, e.g. we use the first-order Escargot prover to reconstruct every inference that Vampire produces.

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). If we remove formulas that were not used in the classical proofs, then 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 non-Class 1 problems, which should be harder for Slakje as it is not complete there.

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

[6] available at http://iltp.de/download/ILTP-v1.1.2-firstorder.tar.gz

GAPT's prover interface, Vampire solves 1938 out of the 2421 non-quantifier-free problems, returning 1585 proofs in the textual TPTP derivation format and 353 satisfiable results (for which Slakje can immediately return non-theorem). Proof replay then produces 1541 resolution proofs, which are converted to 1526 expansion proofs, ultimately yielding 1098 intuitionistic proofs in mLJ. (The remaining 192 theorems are quantifier-free formulas, which we directly passed to the constructivization procedure.) 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 GAPT cannot parse reliably, making it less effective 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 proves 1290 out of 2670 problems with Vampire as a backend. This is significantly more than other state-of-the-art provers such as ileanCoP (proving 813 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 in [9] 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 provers and evaluate its use for proof automation and as a strong reconstruction tactic for hammers in proof assistants based on intuitionistic logic.

Acknowledgements. The author would like to thank the anonymous reviewers for their suggestions which have led to a considerable improvement of this paper. This work has been supported by the Vienna Science and Technology Fund (WWTF) project VRG12-004.

References

1. Baaz, M., Hetzl, S., Weller, D.: On the complexity of proof deskolemization. Journal of Symbolic Logic **77**(2), 669–686 (2012)
2. Berre, D.L., Parrain, A.: The Sat4j library, release 2.2. JSAT **7**(2-3), 59–6 (2010)
3. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development – Coq'Art: The Calculus of Inductive Constructions. Texts in Theoretical Computer Science, Springer (2004)
4. Bibel, W.: Matings in matrices. Communications of the ACM **26**(11), 844–852 (1983)
5. 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)
6. 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)
7. Buss, S.R.: On Herbrand's Theorem. In: Logic and Computational Complexity, Lecture Notes in Computer Science, vol. 960, pp. 195–209. Springer (1995)
8. 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)
9. 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)
10. Czajka, L., Kaliszyk, C.: Hammer for Coq: Automation for dependent type theory. Journal of Automated Reasoning **61**(1-4), 423–453 (2018)
11. Dunchev, C., Leitsch, A., Libal, T., Riener, M., Rukhaia, M., Weller, D., Paleo, B.W.: PROOFTOOL: a GUI for the GAPT framework. In: Kaliszyk, C., Lüth, C. (eds.) Proceedings 10th International Workshop On User Interfaces for Theorem Provers (UITP) 2012. EPTCS, vol. 118, pp. 1–14 (2012)
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., Leitsch, A., Reis, G., Weller, D.: On the generation of quantified lemmas. Journal of Automated Reasoning pp. 1–32 (2018)
15. 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)
16. 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)
17. Herbrand, J.: Recherches sur la théorie de la démonstration. Ph.D. thesis, Université de Paris (1930)

18. Kaliszyk, C., Urban, J.: Learning-assisted automated reasoning with Flyspeck. Journal of Automated Reasoning **53**(2), 173–213 (2014)
19. 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)
20. Maehara, S.: Eine Darstellung der intuitionistischen Logik in der Klassischen. Nagoya Mathematical Journal **7**, 45–64 (1954)
21. McCune, W.: Prover9 and Mace4 (2005–2010), http://www.cs.unm.edu/~mccune/prover9/
22. 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)
23. Miller, D.A.: A compact representation of proofs. Studia Logica **46**(4), 347–370 (1987)
24. 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)
25. Mints, G.: Gentzen-type system and resolution rules. Part II: Propositional logic. In: Logic Colloquium 1990 (1993)
26. 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)
27. Negri, S.: Glivenko sequent classes in the light of structural proof theory. Archive for Mathematical Logic **55**(3-4), 461–473 (2016)
28. Orevkov, V.P.: On Glivenko sequent classes. Trudy Matematicheskogo Instituta imeni V. A. Steklova **98**, 131–154 (1968)
29. 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)
30. 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)
31. 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)
32. Raths, T., Otten, J., Kreitz, C.: The ILTP problem library for intuitionistic logic. Journal of Automated Reasoning **38**(1-3), 261–271 (2007)
33. Reis, G.: Importing SMT and connection proofs as expansion trees. In: Fourth Workshop on Proof eXchange for Theorem Proving, PxTP. pp. 3–10 (2015)
34. 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)
35. 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)

36. The Univalent Foundations Program: Homotopy Type Theory: Univalent Foundations of Mathematics. Institute for Advanced Study (2013), https://homotopytypetheory.org/book
37. Troelstra, A.S., Schwichtenberg, H.: Basic Proof Theory. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press (2000)
38. 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)
39. Voronkov, A.: AVATAR: the architecture for first-order theorem provers. In: Biere, A., Bloem, R. (eds.) 26th International Conference on Computer Aided Verification, CAV 2014. Lecture Notes in Computer Science, vol. 8559, pp. 696–710. Springer (2014)
40. 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)