A Proof-Theoretic Approach to Certifying Skolemization

Kaustuv Chaudhuri, Matteo Manighetti, Dale Miller

A Proof-Theoretic Approach to Certifying Skolemization

Kaustuv Chaudhuri
Inria & LIX, École polytechnique
Palaiseau, France
kaustuv.chaudhuri@inria.fr

Matteo Manighetti
Inria & LIX, École polytechnique
Palaiseau, France
matteo.manighetti@inria.fr

Dale Miller
Inria & LIX, École polytechnique
Palaiseau, France
dale.miller@inria.fr

Abstract

When presented with a formula to prove, most theorem provers for classical first-order logic process that formula following several steps, one of which is commonly called skolemization. That process eliminates quantifier alternation within formulas by extending the language of the underlying logic with new Skolem functions and by instantiating certain quantifiers with terms built using Skolem functions. In this paper, we address the problem of checking (i.e., certifying) proof evidence that involves Skolem terms. Our goal is to do such certification without using the mathematical concepts of model-theoretic semantics (i.e., preservation of satisfiability) and choice principles (i.e., epsilon terms). Instead, our proof checking kernel is an implementation of Gentzen's sequent calculus, which directly supports quantifier alternation by using eigenvariables. We shall describe deskolemization as a mapping from client-side terms, used in proofs generated by theorem provers, into kernel-side terms, used within our proof checking kernel. This mapping which associates skolemized terms to eigenvariables relies on using *outer* skolemization. We also point out that the removal of Skolem terms from a proof is influenced by the polarities given to propositional connectives.

CCS Concepts • **Theory of computation → Proof theory**; *Constraint and logic programming*;

Keywords Skolemization; foundational proof certificates; focusing; sequent calculus; λProlog

1 Introduction

Skolemization [**?**] is a process (of which there are many variants) that removes *strong quantifiers* by instantiating such quantifiers with terms of the form $f(x_1, \ldots, x_n)$ where $n \geq 0$, $x_1, \ldots, x_n$ is a list of distinct *weakly quantified variables*, and f is a *Skolem constant*.[1] Exactly which list of such variables is used depends on which form of skolemization is employed, but, in all cases, the resulting formula contains no strong quantifiers. Theorem provers employ this preprocessing step in part because it removes quantifier alternation: when only weak quantifiers exist, standard first-order unification can be used to discover how all the remaining quantifiers can be instantiated. In particular, forward search strategies such as the inverse method do not need to implement an expensive *eigenvariable condition*.

The correctness of skolemization in first-order classical logic is generally justified by referring to the model theory of classical logic. The main meta-theorem for skolemization is that if the skolemized instance of formula B is satisfiable then the formula B is also satisfiable. Given that this theorem is about satisfiability (and not truth), skolemization is often employed in a *refutation* procedure: if one can demonstrate that the skolemized version of $\neg B$ is unsatisfiable (since, for example, one can derive an empty clause from it), then $\neg B$ is unsatisfiable. Employing the model theory of first-order classical logic again, we know that B is valid and, hence, by completeness we know that B has a proof in a complete proof system such as Gentzen's *LK* sequent calculus [20].

A central issue with skolemization is how to use evidence for the unsatisfiability of a skolemized version of $\neg B$ to formally *certify* that B is a theorem. We are interested in certification in the sense of having proofs formally checked using computerized proof-checkers. One method to achieve this kind of certification is to first formally establish the model-theoretic properties of satisfiability and of equi-satisfiability of skolemization as meta-theorems in a formal reasoning system such as Coq or Isabelle/HOL. Such a meta-theorem would employ significant aspects of the foundations of ordinary mathematics, including axioms of extensionality, infinity, and choice [13]. Certifying B as a theorem would then amount to first checking the evidence for the unsatisfiability of the skolemized version of $\neg B$ (for instance, by checking that a provided refutation is syntactically correct), and then appealing to the model-theoretic meta-theorem to conclude that $\neg B$ is itself unsatisfiable, and hence that B is a theorem.

A more direct and targeted certification can be achieved in theorem provers that contain a choice operator such as Hilbert's ϵ-operator and its associated axioms. Such operators can be used to specify Skolem functions; for instance, the ϵ operator of Isabelle/HOL can be used to justify skolemization [6]. However, this still leaves unsolved the problem of certifying B using proof checkers that do not have such

[1] An occurrence of a quantifier in a formula is *strong* if a cut-free proof that introduces it uses an eigenvariable to instantiate it. Otherwise, it is a *weak* quantifier instance. In some texts our definition of Skolemization is actually considered to be the dual of Skolemization, usually called *Herbrandization*, which keeps the strong quantifiers and replaces weak quantifiers with Skolem functions; the results of this paper can be trivially dualized.

built-in operators, particularly in intuitionistic proof checkers that cannot support such operators (without the use of axiomatic extensions).

1.1 Direct Certification using the Sequent Calculus

In this paper we are interested in a more direct approach: *deskolemizing* the evidence into a proof in a system such as Gentzen's *LK*, which is complete for classical first-order logic without relying on choice operators or foundational axioms. This also avoids the need for powerful proof techniques that would be needed to establish the model-theoretic metatheorems. Instead, one only needs to check that a proposed proof structure does, indeed, describe an *LK* proof.

There are a number of reasons for preferring this certification approach. First, *LK* proofs are easy to import into a variety of other proof systems including higher-order logic and even intuitionistic proof systems. (See, for example, [19, 34] of proof evidence being imported into higher-order proof systems.) But, skolemization is not sound for higher-order logic (without choice) [26] and for intuitionistic logic, so the imported *LK* proofs need to be for the original formulas (without Skolem functions).

Second, an *LK*-proof lets us achieve a high degree of confidence in the correctness of the system. This is not only because of the pedigree of *LK*, but also because it is possible to check *LK* proofs syntactically without appealing to strong axioms such as choice. We can also envision applications that involve interacting with, browsing, and mining formal proof structures [23]. If the proof relies on just *LK*, then the resulting interactions should be rather direct and informative. Choice principles, choice operators, equi-satisfiability, etc. will likely make such interactions more obscure.

1.2 Relating Skolemized Proofs to *LK*

The skolemized form of a formula has only weak quantifiers, so an *LK* proof of this formula would never introduce an eigenvariable. This in turn means that the scopes that are determined by strong quantifiers are not necessarily respected in the *LK* proof of the skolemized formula. To deskolemize is to reorganize the proof in such a way that the strong quantifier scopes can be restored, and then the Skolem terms used to replace such quantifiers can be changed to eigenvariables in order to recover a standard *LK* proof.

Deskolemization has been widely studied for classical first-order logic. In general, deskolemization techniques rely heavily on knowledge of the way the original formula was skolemized. For example, in [25, 26] it was shown that a certain type of skolemization (called *outer* skolemization in Section 2) can be deskolemized in expansion proofs without increasing the size of the expansion proof. A different form of skolemization that is often used in automated theorem provers (called *inner* skolemization in Section 2) was studied in papers such as [3] and [4] where it was shown that eliminating Skolem functions can result in complex and expensive growth of proofs.

1.3 Our Approach to Deskolemization

In this paper we look at a particular class of deskolemization approaches that amount to inferring scopes for strong quantifiers without performing more drastic restructuring of the proof. Briefly, they interpret a given proof evidence involving Skolem functions as an *LK* proof, but distinguish between two different actors involved with proof checking: the *client* is some proof evidence producer (such as a theorem prover) that wants to export checkable proof evidence, while the *kernel* is a program that is entrusted to check proofs in a completely trustworthy fashion, i.e., in terms of *LK*. *Client terms*, the terms that can be used in the client's proof evidence, are allowed to mention Skolem functions, but *kernel terms* use eigenvariables and forbid Skolem functions. In the process of reconstructing an *LK* proof from the client's proof evidence, a dynamically updating client-to-kernel map is maintained that links the Skolem functions used to instantiate strong quantifiers with their corresponding eigenvariables. In our specific setting, the kernel is a logic program and eigenvariables are an abstraction mechanism used by logic programs to hide some of the structure of terms [27]. Since it is impossible for a client to directly refer to such abstractions, the dynamically maintained map is used by the kernel to rewrite the *LK* proof being constructed on the fly.

1.4 Summary of Our Contributions

This paper makes the following contributions to the problem of deskolemizing proof evidence.

1. We provide a modular way to deskolemize proof evidence involving Skolem functions. This modularity is achieved by extending the design of the kernel used in the Foundational Proof Certificate (FPC) framework for defining proof formats [12]. It builds Gentzen-style *LK* sequent calculus proofs using eigenvariables. For outer skolemization proof evidence (defined below), it leads to *LK* proofs free of Skolem functions.
2. We provide a trustworthy implementation of this form of modular deskolemization using the higher-order logic programming language λProlog. Simple inspection of our kernel provides rather immediate confidence that our proof checker only certifies formulas that are, in fact, theorems. One must also trust (in our case) the implementation of λProlog. However, since we are only using the backtracking and higher-order unification features of the logic underlying λProlog, anyone can provide a reimplementation of these features and of our proof checker: in this way, one does not need to trust the particular implementations of λProlog we have used (Teyjus [30] and Elpi [16]).

3. For a skeptic not willing to trust a λProlog style proof checker, we describe how it is possible to cause λProlog to output a fully elaborated and explicit proof certificate that can be checked without the need for unification and backtracking search. In particular, we describe a simple functional program written in Coq that can check such explicit proof certificates. We have also proved that that proof checker is sound, meaning that a successful execution of the checker will cause Coq to accept the corresponding classical logic formula as Coq theorem when classical logic axioms are admitted.
4. We give a precise characterization of the surprising interaction of skolemization and *polarities* arising from focused proofs. It turns out that positive polarities cause problems similar to those of inner skolemization, which is already well known to be difficult to treat syntactically [4, 18]. In either case, the culprit is the ability to suspend processing a connective that would have introduced the eigenvariable (in the unskolemized form) and operate on a different formula that nevertheless uses the eigenvariable by means of its Skolem term, causing leakage of eigenvariables from their scopes.

2 Formulas and Skolemization

We work with the standard language of classical first-order logic. *Terms* ($s, t, \dots$) will, as usual, be built from variables ($x, y, \dots$) and *function applications* of the form $f(t_1, \dots, t_n)$ where f is a *function symbol* of fixed arity n. If the argument list is empty (i.e., if $n = 0$), then we omit the parentheses in function applications. A collection of function symbols together with their arities is called a *signature*; for example, $\{c/0, f/1, g/2\}$. We assume that the set of terms generated from a signature is non-empty (for example, $\{f/1, g/2\}$ is not a signature) and that a symbol is given at most one arity within a signature.

Formulas ($A, B, \dots$) and literals (L) belong to the following grammar:

$$A, B, \dots ::= L \mid A \wedge B \mid \top \mid A \vee B \mid \bot \mid \forall x.\, A \mid \exists x.\, A$$
$$L ::= p \mid \neg p$$

Here, p ranges over *atomic formulas* that are always of the form $a(t_1, \dots, t_n)$ where a is a *predicate symbol* of fixed arity n. As is customary, we shall assume that all formulas are in *negation normal form*: that is, negations have only atomic scope. This normal form is a mild one to assume since the size of a formula and its negation normal form are essentially the same. We write $A^{\perp}$ for the de Morgan dual of A, given by the pairs $p/\neg p$, $\wedge/\vee$, $\top/\bot$ and $\exists/\forall$. We shall also assume that no two occurrences of a quantifier (either $\forall$ or $\exists$) bind variables with the same name; this can always be achieved by α-conversion.

Since we are focused on checking proofs, we shall describe skolemization as a process for replacing universally quantified formulas with Skolem terms. Formally, replacing universal quantifiers in this way is often called *herbrandization* while replacing existential quantifiers usually called *skolemization*. Since the intent of both operations is to ensure that strong quantifiers are removed and that eigenvariables are not used within proofs, it seems unnecessary to introduce a second term and remain with the more commonly used term skolemization.

We shall assume that all first-order formulas for which we perform proof checking contain function symbols and constants from the fixed signature Σ_0. In order to account for skolemization, we introduce another signature, Σ_{sk}, disjoint with Σ_0, whose members are called *Skolem functions*, and which is such that for every arity $n \geq 0$, there are a countably infinite number of members of Σ_{sk} of that arity.

Definition 2.1 (Skolemization)**.** The following standard definitions are from [31].

- An *outer skolemization step* is a pair of formulas in which
  - the first formula, say, B is such that it contains the subformula $\forall x.\, C$ that is not in the scope of any universal quantifier and which is in the scope of existential quantifiers binding the variables $x_1, \dots, x_n$ ($n \geq 0$); and
  - the second formula results from picking an n-arity symbol f from Σ_{sk} that does not appear in B and replacing that occurrence of $\forall x.C$ in B with the instance $[f(x_1, \dots, x_n)/x]C$.
- An *inner skolemization step* is a pair of formulas that is defined analogously with the only difference being that the Skolem term used to instantiate x in C is $f(y_1, \dots, y_m)$ where $y_1, \dots, y_m$ are the free variables of the occurrence of $\forall x.\, C$.
- The formula E is the result of performing *outer skolemization* on B if there is a sequence of outer skolemization steps that carries B to E and where E does not contain any strong quantifiers (i.e., universal quantifiers). Similarly, the formula E is the result of performing *inner skolemization* on B if there is a sequence of inner skolemization steps that carries B to E and where E does not contain any strong quantifiers. □

Note that, necessarily, $m \leq n$ in the two skolemization steps in the definition; moreover, all the variables in the list $y_1, \dots, y_m$ are contained in the list $x_1, \dots, x_n$.

Example 2.2. The formula $\exists x.\, (\neg\, d(x) \vee \forall y.\, d(y))$ can be skolemized as follows.

- Outer: $\exists x.\, (\neg\, d(x) \vee d(f(x)))$
- Inner: $\exists x.\, (\neg\, d(x) \vee d(f))$

Note that an *LK* proof of the outer skolemized form would require a contraction and two witness terms, c and $f(c)$

(for some constant c), just like the *LK* proof of the original unskolemized formula. The inner skolemized form, on the other hand, has a simple *LK* proof that provides the witness f for x and does not require a contraction. □

The main result about skolemization is the following theorem. Its proof can be found in a number of textbooks and papers: see, in particular, [2] and [33, Section 4.5].

Theorem 2.3. *Let B be a first-order formula over the signature Σ_0 and let E be either an inner or outer skolemization of B. If E is satisfiable then so is B.* □

Proofs involving inner skolemization have always been more problematic, and no purely syntactical treatment exists for them. Therefore, in the following sections we will only be treating outer skolemizations, and return to the topic of inner skolemization in Section 7.

3 Focused Sequent Calculus

We argued in Section 1.1 that our view of *certification* was founded on building explicit sequent calculus proofs. This certification process can be viewed as a kind of protocol between two agents. One agent is the *client*, who has constructed some evidence such as a resolution refutation or an expansion proof. The other agent is the *proof-checker*, which we also call the *kernel*, which is a trusted implementation of a particular proof system such as the *LK* sequent calculus. The client needs to convince the kernel of the veracity of its evidence, so it will have to guide the kernel towards building a complete sequent proof. Note that there is no need to *store* the proof that the kernel builds – it is enough that the kernel *performs* it.

Given this description of the certification process, it is immediately apparent that employing the original *LK* sequent calculus of Gentzen is problematic. The main issue is the amount of information the client must provide to guide the construction of an *LK* proof. Nearly every sequent can be the conclusion of a structural rule (weakening and contraction), a cut rule, and a (possibly large) number of introduction rules for all the formulas in the sequent. And, once the client instructs the kernel to attempt one such inference rule, its corresponding premises will then need to be guided in a similar way.

Fortunately, not every choice in building a proof is the same. Some choices are important because they introduce fresh information—such as witness terms—into the proof and because making the wrong choice can cause a failed proof attempt. Other choices are unimportant: for instance, the choice of the name of an eigenvariable or the order in which conjunctive branches are proved cannot possibly break a proof attempt. A careful study of such choices in the proof leads us to *polarities* and *focusing*, two recent advances in the proof theory of the sequent calculus (and several related formalisms). First developed for sequent calculi for linear

Asynchronous rules

$$\frac{\Sigma \vdash \Gamma \Uparrow A, \Theta \quad \Sigma \vdash \Gamma \Uparrow B, \Theta}{\Sigma \vdash \Gamma \Uparrow A \mathbin{\bar{\wedge}} B, \Theta} \qquad \frac{}{\Sigma \vdash \Gamma \Uparrow \bar{\top}, \Theta} \qquad \frac{\Sigma \vdash \Gamma \Uparrow A, B, \Theta}{\Sigma \vdash \Gamma \Uparrow A \mathbin{\bar{\vee}} B, \Theta}$$

$$\frac{\Sigma \vdash \Gamma \Uparrow \Theta}{\Sigma \vdash \Gamma \Uparrow \bar{\bot}, \Theta} \qquad \frac{\Sigma, y \vdash \Gamma \Uparrow [y/x]A, \Theta}{\Sigma \vdash \Gamma \Uparrow \forall x.\, A, \Theta}\ y \notin \Sigma$$

Synchronous rules

$$\frac{\Sigma \vdash \Gamma \Downarrow A \quad \Sigma \vdash \Gamma \Downarrow B}{\Sigma \vdash \Gamma \Downarrow A \mathbin{\hat{\wedge}} B} \qquad \frac{}{\Sigma \vdash \Gamma \Downarrow \hat{\top}} \qquad \frac{\Sigma \vdash \Gamma \Downarrow A}{\Sigma \vdash \Gamma \Downarrow A \mathbin{\hat{\vee}} B} \qquad \frac{\Sigma \vdash \Gamma \Downarrow B}{\Sigma \vdash \Gamma \Downarrow A \mathbin{\hat{\vee}} B}$$

$$\frac{\Sigma \vdash (\mathtt{wf}\ t) \quad \Sigma \vdash \Gamma \Downarrow [t/x]A}{\Sigma \vdash \Gamma \Downarrow \exists x.\, A}$$

Identity rules

$$\frac{}{\Sigma \vdash \Gamma, \neg p \Downarrow p}\ \text{init} \qquad \frac{\Sigma \vdash \Gamma \Uparrow A \quad \Sigma \vdash \Gamma \Uparrow A^{\perp}}{\Sigma \vdash \Gamma \Uparrow \cdot}\ \text{cut}$$

Structural rules

$$\frac{\Sigma \vdash \Gamma, R \Uparrow \Theta}{\Sigma \vdash \Gamma \Uparrow R, \Theta}\ \text{store} \qquad \frac{\Sigma \vdash \Gamma, P \Downarrow P}{\Sigma \vdash \Gamma, P \Uparrow \cdot}\ \text{decide} \qquad \frac{\Sigma \vdash \Gamma \Uparrow N}{\Sigma \vdash \Gamma \Downarrow N}\ \text{release}$$

In the store rule, R is a positive formula or a literal

Figure 1. Rules of *LKF*. Γ is a multiset of positive formulas or literals, and Θ is a list of formulas.

logic [1, 21] and then extended to a wide variety logics and proof systems, focusing can be seen as a way of organizing proofs in such a way that choice points are minimized. Moreover, judicious use of polarities allows a general proof system to mimic a wide spectrum of other proof systems. Thus, focused proofs form the basis of the *foundational proof certificate* framework, where the kernel is based on a focused variant of *LK* known as *LKF* [12, 24].

Formulas in *LKF* are like those of *LK*, but the formulas are divided into two polarities, *positive* ($P, Q, \ldots$) and *negative* ($N, M, \ldots$), that we explain further below. The notion of duals is extended from the unpolarized case with the pairs $\hat{\wedge}/\bar{\vee}$, $\hat{\top}/\bar{\bot}$, $\hat{\vee}/\bar{\wedge}$, and $\hat{\bot}/\bar{\top}$.

$$A, B, \ldots ::= P \mid N \qquad \text{(formulas)}$$

$$P, Q, \ldots ::= p \mid A \mathbin{\hat{\wedge}} B \mid \hat{\top} \mid A \mathbin{\hat{\vee}} B \mid \hat{\bot} \mid \exists x.\, A \qquad \text{(positive formulas)}$$

$$N, M, \ldots ::= \neg p \mid A \mathbin{\bar{\wedge}} B \mid \bar{\top} \mid A \mathbin{\bar{\vee}} B \mid \bar{\bot} \mid \forall x.\, A \qquad \text{(negative formulas)}$$

For the propositional connectives, the polarity amounts to an annotation on the connective (written with a superposed + or −); quantifiers and literals, on the other hand, have a unique polarity. The polarized versions of the propositional connectives are equivalent: $A \mathbin{\hat{\wedge}} B$ and $A \mathbin{\bar{\wedge}} B$ are not only equi-provable, but each implies the other (this is a consequence of Theorem **??**). However, positive and negative formulas have very different proofs, both in size and in shape.

Intuitively, the introduction rules for negative formulas are *invertible*: that is, these rules have the property that their collection of premises are *equivalent* to their conclusions.

Thus, the order in which these rules are applied is irrelevant and does not need to be communicated by the client; we say that the kernel works *asynchronously*. For instance, the rules for $\bar{\wedge}$ and $\bar{\vee}$ are the following (modulo minor differences):

$$\frac{\vdash A, \Delta \quad \vdash B, \Delta}{\vdash A \mathbin{\bar{\wedge}} B, \Delta} \qquad \frac{\vdash A, B, \Delta}{\vdash A \mathbin{\bar{\vee}} B, \Delta}$$

A positive (non-atomic) formula, on the other hand, has inference rules that are not necessarily invertible, meaning that its introduction rule may involve a choice and its premise(s) may not be equivalent to its conclusion. Applying such a rule involves an essential choice that must be communicated by the client, so we say that the kernel works *synchronously*. For $\dot{\vee}$, for instance, the synchronous rules are:

$$\frac{\vdash A, \Delta}{\vdash A \mathbin{\dot{\vee}} B, \Delta} \qquad \frac{\vdash B, \Delta}{\vdash A \mathbin{\dot{\vee}} B, \Delta}$$

These rules encode an essential choice between the two operands A and B. The two polarized variants of $\vee$ can equivalently be seen as encoding two separate kinds of choice: *internal* (i.e., made by the kernel) and *external* (communicated to the kernel).

Following a technique pioneered by Andreoli [1], we separate the two kinds of inference rules by using the following two kinds of sequents:

$\Sigma \vdash \Gamma \Downarrow A$	synchronous with A *under focus*
$\Sigma \vdash \Gamma \Uparrow \Theta$	asynchronous sequent

The *context* Γ, called the *store*, is a multiset of positive formulas or literals, and Θ, called the *asynchronous zone*, is a *list* of formulas. Σ is the *signature*, which not only contains the arities of the function symbols as before but also includes the set of *eigenvariables* that can be free in the terms to the right of $\vdash$. We say that a term t is well-formed in Σ, written $\Sigma \vdash (\mathtt{wf}\ t)$ to mean that all the function symbols in t are used with the correct arities defined in Σ, and that all the free variables of t are contained in the set of eigenvariables in Σ.

The full list of inference rules for *LKF* is in Figure 1. A proof in *LKF* can be seen as an alternation of two kinds of *phases*, reading the rules from conclusion to premises. The *synchronous phase* starts with a sequent of the form $\Sigma \vdash \Gamma \Uparrow \cdot$ as conclusion; a positive formula is chosen for *focus* using the decide rule and within the entire phase the focused formula is required to be the formula introduced. The client needs to communicate all the choices and witness terms made during the synchronous phase to the kernel. The synchronous phase ends with the init rule when the focused formula is an atom (and the client may need to tell the kernel which is the dual literal), or may transition to the *asynchronous phase* with the release rule that is applicable when the focus is a negative formula. Note that in the init rule if the dual of the focused formula is not in the context then the proof attempt is considered a *proof attempt failure* since there is no other inference rule available to prove a focus on a positive literal; if this happens, the kernel may try to backtrack over other essential choices in the same or an earlier synchronous phase of search. In the asynchronous phase a rule is applied to the leftmost formula in the asynchronous zone; if it is a positive formula or a literal, it is stored, and in every other case an asynchronous rule is used to decompose this formula. Finally, when the asynchronous zone is empty, i.e., when we are back to a sequent of the form $\Sigma \vdash \Gamma \Uparrow \cdot$, then the cycle begins anew.

Let B be an unpolarized formula and let $\hat{B}$ be a polarized formula that results from placing either a $+$ or $-$ superscript on every connective and constant where allowed. We shall also assume that atomic formulas are polarized arbitrarily: they could be all negative, all positive, or some mixture of these two, and the occurrences of $\neg$ are adjusted accordingly. The following theorem is proved in [24].

Theorem 3.1 (Soundness and Completeness)**.** *Let B be a formula of first-order classical logic. If B is a theorem, then $\cdot \vdash \cdot \Uparrow \hat{B}$ is derivable for every polarized version $\hat{B}$ of B. Furthermore, if $\cdot \vdash \cdot \Uparrow \hat{B}$ is provable for some polarized version $\hat{B}$ of B, then B is a theorem.* □

4 Augmented *LKF* and Foundational Proof Certificates

In this section we will describe how we use the *LKF* system to build a protocol for mediating the communications between a client, who already has some proof evidence in hand, and the kernel, (a.k.a. the proof checker). This protocol is the basis for the *foundational proof certificates* framework [12]. The key idea is to augment the *LKF* proof system as follows.

- A *proof certificate* is threaded through every sequent and inference rule: these certificates are term structures that contain the client's proof evidence.
- Additional premises are added to the *LKF* inference rules: these premises manipulate and extract information from proof certificates.

There are two kinds of additional premises added to inference rules. The first kind, the *clerks*, are added to asynchronous rules: clerks perform routine maintenance of proof certificate information. The second kind, the *experts*, are added to synchronous rules and they are responsible for attempting to find important information within the proof certificate to guide the possible choices of the kernel. For instance an expert may inform the kernel which of the two rules to use for $\dot{\vee}$-introduction or which witness term to use for $\exists$-introduction.

The augmented version of *LKF*, called LKF^a, uses the following kinds of sequents.

$\Xi; \Sigma \vdash \Gamma \Downarrow A$	synchronous with A *under focus*
$\Xi; \Sigma \vdash \Gamma \Uparrow \Theta$	asynchronous

Here, Ξ stands for a *proof certificate*, which is explained in more detail below; note, however, that certificates do not affect the meaning of a sequent, and hence are a passive and

Asynchronous rules

$$\frac{\Xi_1; \Sigma \vdash \Gamma \Uparrow A, \Theta \quad \Xi_2; \Sigma \vdash \Gamma \Uparrow B, \Theta \quad \wedge_c(\Xi_0, \Xi_1, \Xi_2)}{\Xi_0; \Sigma \vdash \Gamma \Uparrow A \mathbin{\bar{\wedge}} B, \Theta}$$

$$\frac{}{\Xi_0; \Sigma \vdash \Gamma \Uparrow \bar{\top}, \Theta}$$

$$\frac{\Xi_1; \Sigma \vdash \Gamma \Uparrow A, B, \Theta \quad \vee_c(\Xi_0, \Xi_1)}{\Xi_0; \Sigma \vdash \Gamma \Uparrow A \mathbin{\bar{\vee}} B, \Theta} \qquad \frac{\Xi_1; \Sigma \vdash \Gamma \Uparrow \Theta \quad \bot_c(\Xi_0, \Xi_1)}{\Xi_0; \Sigma \vdash \Gamma \Uparrow \bar{\bot}, \Theta}$$

$$\frac{\Xi_1; \Sigma, (\mathsf{copy}\ t\ y) \vdash \Gamma \Uparrow [y/x]A, \Theta \quad \forall_c(\Xi_0, \Xi_1, t)}{\Xi_0; \Sigma \vdash \Gamma \Uparrow \forall x.\, A, \Theta}\ y \notin \Sigma$$

Synchronous rules

$$\frac{\Xi_1; \Sigma \vdash \Gamma \Downarrow A \quad \Xi_2; \Sigma \vdash \Gamma \Downarrow B \quad \wedge_e(\Xi_0, \Xi_1, \Xi_2)}{\Xi_0; \Sigma \vdash \Gamma \Downarrow A \mathbin{\hat{\wedge}} B} \qquad \frac{\top_e(\Xi_0)}{\Xi_0; \Sigma \vdash \Gamma \Downarrow \hat{\top}}$$

$$\frac{\Xi_1; \Sigma \vdash \Gamma \Downarrow A_i \quad \vee_e(\Xi_0, \Xi_1, i)}{\Xi_0; \Sigma \vdash \Gamma \Downarrow A_1 \mathbin{\hat{\vee}} A_2}\ i \in \{1, 2\}$$

$$\frac{\Sigma \vdash (\mathsf{copy}\ t\ s) \quad \Xi_1; \Sigma \vdash \Gamma \Downarrow [s/x]A \quad \exists_e(\Xi_0, \Xi_1, t)}{\Xi_0; \Sigma \vdash \Gamma \Downarrow \exists x.\, A}$$

Identity rules

$$\frac{\mathrm{init}_e(\Xi_0, l)}{\Xi_0; \Sigma \vdash \Gamma, l\colon \neg p \Downarrow p}\ \text{init}$$

$$\frac{\Xi_1; \Sigma \vdash \Gamma \Uparrow A \quad \Xi_2; \Sigma \vdash \Gamma \Uparrow A^{\perp} \quad \mathrm{cut}_e(\Xi_0, \Xi_1, \Xi_2, A)}{\Xi_0; \Sigma \vdash \Gamma \Uparrow \cdot}\ \text{cut}$$

Structural rules

$$\frac{\Xi_1; \Sigma \vdash \Gamma, l\colon P \Downarrow P \quad \mathrm{decide}_e(\Xi_0, \Xi_1, l)}{\Xi_0; \Sigma \vdash \Gamma, l\colon P \Uparrow \cdot}\ \text{decide}$$

$$\frac{\Xi_1; \Sigma \vdash \Gamma \Uparrow N \quad \mathrm{release}_e(\Xi_0, \Xi_1)}{\Xi_0; \Sigma \vdash \Gamma \Downarrow N}\ \text{release}$$

$$\frac{\Xi_1; \Sigma \vdash \Gamma, l\colon R \Uparrow \Theta \quad \mathrm{store}_c(\Xi_0, \Xi_1, l)}{\Xi_0; \Sigma \vdash \Gamma \Uparrow R, \Theta}\ \text{store}$$

In the store rule, R is a positive formula or a literal

Figure 2. Rules of LKF^a, an augmented version of *LKF*. Γ is a multiset of pairs of the form l:R where l is an index and R is a positive formula or literal, and Θ is a list of formulas.

abstract participant from a logical perspective. Their sole purpose will be in guiding the construction of LKF^a proofs. Both of the structures Σ and Γ are generalized in LKF^a over what they were in *LKF*. In particular, Σ is now more than a signature: it is a set of pairings of the form $(\mathsf{copy}\ t\ y)$ where t is a client-side term (containing, for example, Skolem functions) that is associated to the eigenvariable y (that is, a kernel-side term). In a similar fashion, the context Γ is extended to be a set of pairs of the form l:R where l is an *index* and R is a positive formula or a literal. The exact structure of indexes and client terms are not specified by the kernel but are a detail provided by the definition of a proof certificate format. The context Θ is as before in *LKF*.

There are several important things to observe about the LKF^a calculus shown in Figure 2. First, predicates with subscript e are experts and those with subscript c are clerks. We drop the explicit reference to the polarity of clerks and experts since these can be inferred easily: e.g., we write $\wedge_c$ instead of $\bar{\wedge}_c$ since clerks are defined only for negative connectives. Second, the first argument to the expert or clerk is always the proof certificate of the conclusion, and can be interpreted as an input. The other proof certificate arguments can be interpreted as outputs yielding the continuation proof certificates for the premises (if any). Additional arguments may be indexes (in the case of init_e, decide_e, and store_c), a client-side name to associate with an eigenvariable (in the case of $\forall_c$), rule selectors (in the case of $\vee_e$), witness terms (in the case of $\exists_e$), or formulas (in the case of cut_e).

Specifications and implementations of previous versions of proof checkers for the Foundational Proof Certificate framework [7, 11, 12] did not address the fact that client-side terms might be different than kernel-side terms. Substitution terms are not always part of some particular presentation of proof evidence, since unification during proof checking can reconstruct such substitutions, so the difference between client-side and kernel-side terms does not always need to be addressed in proof checkers. As we have seen, however, there can be significant differences between these two classes of terms. We now describe how to extend the previous approach of FPC-based checkers to account for that difference.

The predicate $(\mathsf{copy}\ \cdot\ \cdot)$ in the LKF^a proof system can be formally defined using *copy-clauses*, a standard technique used to encode both term-level equality and substitutions in logic programming [28]. The copy-clauses based on the signature $\{a/0, f/1, g/2\}$ have the following λProlog specification. (We do not assume any advance knowledge of λProlog: for more information about that language, see [29].)

```
copy a a.
copy (f X)   (f U)   :- copy X U.
copy (g X Y) (g U V) :- copy X U, copy Y V.
```

It is easy to show that if t and s are two closed terms over the signature $\{a/0, f/1, g/2\}$, then `(copy t s)` is provable from these clauses if and only if $t = s$. Obviously, any arbitrary first-order signature can be translated into such a set of copy-clauses: in particular, if Σ is such a first-order signature then we write $C(\Sigma)$ to denote the set of copy-clauses determined by that signature.

The inference rules in Figure 2 can be implemented directly in λProlog, as has been described in several other papers [7, 11, 12]. Although such implementations can be small, we present here only a few clauses. First, two simple clauses for the two disjunctions:

```
async Cert ((A or- B)::R) :- orC Cert Cert',
                             async Cert' (A::B::R).
sync  Cert (A or+ B) :- orE Cert Cert' C,
                        ((C = left,  sync Cert' A);
                         (C = right, sync Cert' B)).
```

Here, the sequents $\Xi; \Sigma \vdash \Gamma \Uparrow \Theta$ and $\Xi; \Sigma \vdash \Gamma \Downarrow A$ are represented by the atomic formulas `(async Cert Theta)` and `(sync Cert A)`, respectively: the encoding of Σ and Γ are captured by features found in the (intuitionistic) logic underlying λProlog. Thus, the two clauses above implement the intended meaning of the focused introduction rules for $\bar{\forall}$ and $\hat{\forall}$, respectively.

The introduction rules for the quantifiers employ the copy-clauses to translate client-side terms to kernel-side terms. In particular, consider the following two λProlog clauses specifying the introduction of the quantifiers.

```
async Cert ((all B)::R) :- allCx Cert Cert' T,
      pi w\ copy T w => async Cert' ((B w)::R).
sync  Cert (some B)  :- someE Cert Cert' T,
      copy T S, sync Cert' (B S).
```

The universal quantification of λProlog (`pi w\`) implements the eigenvariable feature needed for the LKF^a proof system and the implication `=>` is used to extend the program clauses for `copy` with a new atomic clause `(copy T w)`, which is only usable within the scope of `pi w\`. In this way, the Σ context in Figure 2 is implemented via λProlog's intuitionistic context.

The copy-clauses can now be used uniformly to perform *deskolemization* in the following sense. Assume that both the kernel and client agree on the signature Σ_0 and that the copy-clauses $C(\Sigma_0)$ derived from that signature are added to the kernel specification. During proof checking, new atomic copy-formulas are added to the Σ context whenever a strong quantifier is encountered (via the first clause displayed above). Whenever the client computes (via the existential expert `someE`) a client-side term `T`, it is then translated to the kernel-side formula `S` by the query `copy T S`.

Example 4.1. Assume that the base signature for both the client and the kernel is $\Sigma = \{a/0, f/1, g/2\}$. Also assume that the client is using $h/1$ as a Skolem function and that the kernel has introduced two eigenvariables x and y and that Γ contains exactly the two associations `(copy (h a) x)` and `(copy (h (f a)) y)`. The λProlog query

$$C(\Sigma), \Gamma \vdash \texttt{(copy (g (h (f a)) (f (h a))) X)}$$

for some logic variable `X` will have a unique solution, namely, the one that binds `X` to `(g y (f x))`. It is this step that performs deskolemization. Note, however, that we do not necessarily assume that deskolemization is determinate. In particular, if the Γ context contained the atoms `(copy (h a) x)` and `(copy (h a) y)`, then there are two solutions to the query `(copy (g (h a) (f a)) X)`, namely, binding `X` to either `(g x (f a))` or `(g y (f a))`. □

Nondeterminism in deskolemization is not a soundness problem in the context of the kernel we have described here: instead, this nondeterminism may cause the kernel to backtrack and to examine more than one deskolemization in order to finish proof checking.

Observe that given an LKF^a sequent, we can easily obtain a corresponding LKF sequent by removing the proof certificate, replacing every instance of `(copy` t x`)` in the signature with `(wf` x`)`, and dropping the indexes on the formulas in the store. Call this its *underlying sequent*. The following property is proved by a simple structural induction on LKF^a proofs.

Theorem 4.2 (Soundness of LKF^a)**.** *If an LKF^a sequent is derivable, then its underlying sequent is derivable in LKF and the unpolarized version of that sequent is provable in LK.* □

It is important to note that LKF^a is sound by construction: no specification for the clerks and experts provided by the client can lead the kernel to prove a non-theorem. Such a strong soundness property is a critical feature of a proof checking kernel.

What is formally called an FPC is a collection of type declarations describing the constructors for certificates and indexes and a collection of clauses specifying the clerk and expert relations. Once these collections are added to the λProlog specification of the inference rules in Figure 2, one has a proof checker that will check one particular format of proof certificates. Many such formats have been so defined using FPCs: these include resolution refutations, sequent calculus proofs, expansion trees, Frege proofs, and rewriting proofs [9, 10, 12]. The notion of formulas and terms within the kernel may both be different from those notions used by the client. Polarization then becomes a mapping from client-side to kernel-side formulas. Likewise, deskolemization is a mapping from client-side to kernel-side terms.

We can state a kind of completeness theorem for how skolemized proof evidence can be used as proof evidence for the original unskolemized theorems. Assume that B is a closed formula and let C be the result of applying outer skolemization to B. Also assume that we are given an FPC, $\mathcal{P}$, that polarizes all occurrences of propositional connectives negatively and that defines proof checking for skolemized proof evidence with a skolemized theorem. Thus, we can assume that this FPC does not need to define the experts $\wedge_e$, $\vee_e$, and $\top_e$ (since the positive propositional connectives do not appear) as well as the clerk $\forall_c$ (since a skolemized formula has no strong quantifiers). Finally, let $\mathcal{P}'$ be the FPC that results from adding to $\mathcal{P}$ the following clause.

```
allCx  Cert Cert T.
```

This specification of `allCx` indicates that some term—unspecified at this point—will name the eigenvariable used to encode the universal right-introduction rule: the association between that term `T` and eigenvariable, say `w`, is made by the λProlog assumption `(copy T w)` (as described above). Given that these various assumptions hold, then we can prove the following: if it is checkable that the certificate Ξ satisfies the FPC $\mathcal{P}$ as a proof of C then the certificate Ξ satisfies the FPC $\mathcal{P}'$ as a proof of B. Thus, if the client satisfies two major requirements on proof evidence—namely, that propositional

connectives are polarized negatively and that skolemization is the outer variety—then the same skolemized proof evidence used with a skolemized formula can immediately be seen as proof evidence of the unskolemized theorem.

5 Experiments with an Implementation

We have implemented the proof checking kernel described in this paper and have conducted several experiments with it. The full code can be found at the following Github repository: [**?**]. It has been trivial to incorporate previous FPCs (those that assumed that client-side and kernel-side terms coincide) to execute on this extended proof checker. One immediate experiment consists of transforming *LK* proofs of skolemized end-sequents to *LK* (via LKF^a) proofs of the original (unskolemized) formulas. (Here, we are assuming that the right introduction rules for disjunction and conjunction are the invertible rules since these match directly their negatively biased versions.) The repository contains two additional and more significant examples. One involves simple reasoning using geometric formulas: in that setting, Skolem terms are used in a rather natural and familiar fashion. In the rest of this section, we describe the other example provided since it is more involved and universal in its scope.

Expansion trees [26] are a proof formalism that generalizes the notion of Herbrand disjunctions to formulas with arbitrary quantifiers (and to formulas with higher-order quantification). There are also two variations of expansion trees: one using *select variables* to instantiate strong quantifiers and one using Skolem terms to instantiate strong quantifiers. We have implemented three procedures for checking different kinds of proof evidence based on this formalism: one for expansion trees with select variables, one replacing select variables with Skolem terms, and one for expansion trees of skolemized formulas (thus, containing neither Skolem terms nor select variables).

Expansion trees such as those we will now describe are used, in fact, in the deskolemization procedure of [4], implemented in the GAPT system [17].

5.1 Expansion Trees with Select Variables

As we described in Section 2, we assume that formulas are in negation normal form.

Definition 5.1 (Expansion trees)**.**

- A literal or logical constant is an expansion tree for itself.
- If Q_1 and Q_2 are expansion trees of A_1 and A_2, then $(eOr\ Q1\ Q2)$ and $(eAnd\ Q1\ Q2)$ are expansion trees for $A_1 \vee A_2$ and $A_1 \wedge A_2$ respectively
- If u is a variable (called a *select variable*) and Q is an expansion tree of $[u/x]A$, then $(eAll\ u\ Q)$ is an expansion tree for $\forall x.\, A$.

```
kind et                type.
type eTrue, eFalse     et.
type eLit              et.
type eAnd, eOr         et -> et -> et.
type eAll              i  -> et -> et.
type eSome             list (pair i et) -> et.
```

Figure 3. The datatype for expansion trees. The `kind` declaration introduces a primitive type `et` and the `type` declarations introduces constructors for this primitive type.

```
kind address       type.
type root          address.
type lf, rg, dn    address -> address.
type idx           address -> index.
typeabbrev context list (pair address et).
type astate        context -> context -> cert.
type dstate        context -> context -> cert.
type sstate        context -> pair address et -> cert.
```

Figure 4. Certificate constructors for expansion trees. The primitive types `index` and `cert` are declared as part of the kernel. The type `address` is introduced for this particular FPC.

```
orC    (astate Left ((pr Add (eOr E1 E2))::Qs))
       (astate Left ((pr (lf Add) E1)::
                     (pr (rg Add) E2)::Qs)).
andC   (astate Left ((pr Add (eAnd E1 E2))::Qs))
       (astate Left ((pr (lf Add) E1)::Qs))
       (astate Left ((pr (rg Add) E2)::Qs)).
someE  (sstate Left
               (pr Add (eSome ((pr Term ET)::nil)))
       (dstate Left ((pr (dn Add) ET)::nil)) Term.
allCx  (eAll Term Cert) Cert Term.
```

Figure 5. Some of the clerks and experts for expansion trees. All of these λProlog clauses are simply atomic formulas that perform some pattern matching and simple transformations on certificates.

- If $t_1, \ldots, t_n$ is a list of *expansion* terms and if Q_i is an expansion tree for $[t_i/x]A$ (for $i \in 1..n$), then

$$(eSome\ [(t_1, Q_1), \ldots, (t_n, Q_n)])$$

  is an expansion tree for $\exists x.\, A$. □

Expansion terms can certainly contain select variables. The formal, stand-alone definition of expansion trees requires additional correctness conditions to be assumed (that a certain propositional formula derived from the expansion tree is a tautology and that a certain relationship on select variables is acyclic) but these conditions are not needed here since they will be replaced by the proof checking kernel itself. Select variables within expansion trees are rather similar to Skolem terms: select variables can be seen as nothing but another mechanism for naming eigenvariables, in the spirit of client vs. kernel terms.

The datatype for expansion trees can be formalized by the λProlog signature in Figure 3 and the more general notion

of certificate based on expansion trees is given in Figure 4. There, proof certificates (terms of type `cert`) are built from three constructors: `astate` is consumed during the asynchronous phase and records two contexts representing some information about the storage zone Γ and the asynchronous zone Θ; `sstate` is consumed during the synchronous phase and records the storage and the formula under focus; and `dstate` is used to break focusing on adjacent existential introductions. Formulas are paired in the certificate with the expansion trees to which they are associated. Addresses are essentially paths through the proposed theorem: they are used to uniquely describe subformulas. For example, such addresses are used to link stored formulas (note that indexes contain addresses) with expansion trees sorted within certificate terms.

The main clerks and experts are specified in Figure 5. Since connectives are polarized negatively, most of the work is carried out by clerks that simply consume expansion trees and reorganize internal components of certificates. When proof checking encounters a strong quantifier, the expansion-tree-cum-certificate contains the select variable associated to it: we then use the `allCx` to instruct the kernel to create a new eigenvariable and associate the client's select variable as a name for that eigenvariable. When proof checking meets an existential node, together with the list of terms by which the existential should be instantiated, we can simply communicate one of the client's expansion terms to the kernel which then proceed to translate it to a kernel term. Note that, in the code, we make the assumption that only one term is present in the list: this is because contraction is handled by the expert for the decide rule (not shown here).

Note that the mechanism we have described as deskolemization is exactly the same mechanism that can replace variable names (select variables) with eigenvariables. Note also that if the expansion tree that is being checked uses a select variable more than once to name different eigenvariables, the checker will need to deal with nondeterminism in sorting out which assignment of select variable to eigenvariable leads to a proper proof. Similar to the comment in Example 4.1, such non-unique naming is not a soundness problem: it can, however, raise the cost and complexity of proof checking.

5.2 Skolem Expansion Trees

Skolem expansion trees [26] are essentially the same as expansion trees except that select variables are replaced by Skolem terms. It turns out that the FPC (given in Figures 3, 4, and 5) for regular expansion trees works without change in the setting where select variables are replaced by Skolem constants. In a sense, Skolem terms act as names in the same way as select variables acted as names of eigenvariables. Critical to the perspective that Skolem terms and select variables act as names is the fact that the copy clauses used within the kernel are never extended to copy a select variable or a Skolem function themselves. In particular, it is important that copy clauses do not treat Skolem functions in the same way as function symbols in the basic signature Σ_0.

5.3 Expansion Trees of Skolemized Formulas

We now turn our attention to the setting where the client has an expansion tree relative to a skolemized formula but we would like to use it as proof evidence of the original, unskolemized formula. In this case, since there are no strong quantifiers left in the skolemized formula, the expansion tree will not contain any select variables (nor any Skolem terms). Accordingly, we modify the `allCx` clerk to be the clause we introduced at the end of Section 4.

```
allCx Cert Cert T.
```

Thus, when the checker finds a strong quantifier it will simply associate to the newly created eigenvariable a logic variable (here, `T`) as the name for it. This variable will ultimately be instantiated to be an actual Skolem term (through the interaction of proof checking and unification).

6 Towards a Coq-Based Proof Checker

The mechanism outlined so far uses an implementation of the FPC framework in λProlog. By default, the checker so described *performs* an *LK* proof with on-the-fly replacement of Skolem terms by eigenvariables, but this *LK* proof is not itself recorded. However, it is easy to instrument the checker to record the proof; indeed, it does not even require any modification to the kernel or the FPCs. Instead, we embed the checker inside a *pairing* checker that not only *consumes* the original proof evidence but also *produces* a fully explicit proof certificate that closely mirrors the LKF^a derivation, called a *max cert*. Crucially, this pairing technique requires no modifications whatsoever in the implementation of the LKF^a kernel; it is handled entirely by the experts and clerks. The details of this technique can be found in [7].

Such a fully explicit proof can, in principle, be checked by simple, deterministic, and even *certified* proof checkers. We have started to implement just such a checker in Coq. This checker is given as input an *LKF* formula and a max cert, which records all choices and witness terms in the *LKF* proof in a higher-order tree-like data structure. To check this proof we simply need to implement an interpreter for *LKF* proofs that proceeds by structural recursion on the certificate. The interpreter is defined as a recursive fixed point that computes either `True` or `False`, depending on whether the certificate is accepted or not. To check the proof in Coq we merely need to apply this fixed point computation to the representation of the input proof and see that the normal form is `True`.

The implementation of this recursive checker amounts to about 130 lines of definition text, and a small but varying amount of additional definitions to encode the signature. This checker implementation would ideally be equipped with a formal meta-theorem that asserts that if the check succeeds

than there is an interpretation of the *LKF* end-sequent as a (classically) true `Prop` of Coq. We have completed this certification for the propositional subset of *LKF* and are in the process of extending it to the significantly more involved case involving quantifiers that we will now describe. The Coq code can be found in the `coq/` subdirectory of the repository mentioned at the beginning of Sec. 5.

The main reason the quantifiers are complicated is the issue of exporting the max cert *LKF* proof in terms of eigenvariables from our version of LKF^a in Sec. 4. As should be clear from the asynchronous rule for $\forall$ in Fig. 2, the continuation certificate Ξ_1 in the premise is not abstracted over the eigenvariable y; rather, it contains many occurrences of the Skolem term t which may be copied to y. However, the mechanism for copying uses full backtracking search in λProlog in terms of the `copy` predicate, which is not exportable to a deterministic checker as the one we would define in Coq.

Thus we need to be able to transform the certificate in the λProlog level to make the abstraction over the eigenvariable explicit in the structure of the certificate. This requires a variant of the LKF^a kernel with a $\forall$ rule that looks as follows:

$$\frac{\begin{array}{c}\forall_c(\Xi_0, \Xi_1, t) \qquad \Sigma, (\mathsf{copy}\ t\ y) \vdash (\mathsf{copy}\ \Xi_1\ (\Xi_2\ y)) \\ (\Xi_2\ y); \Sigma, (\mathsf{copy}\ y\ y) \vdash \Gamma \Uparrow [y/x]A, \Theta\end{array}}{\Xi_0; \Sigma \vdash \Gamma \Uparrow \forall x.\, A, \Theta}\ y \notin \Sigma$$

The second premise here shows how to transform a certificate Ξ_1 with occurrences of a Skolem term t into a certificate Ξ_2 abstracted over an eigenvariable, which is proved by a derivation in the same extended $(\mathsf{copy}\ \cdot\ \cdot)$ context in which the original rule was applied. The third premise then continues with the eigenvariable version of the certificate under a trivial assumption about copying the eigenvariable to itself.

There are a few non-trivial ramifications of this transition from copying not only terms, which have a fixed structure (even though they have some parametric elements such as the signature of constants and predicate symbols), but also certificates, which do not have a fixed structure. Indeed, different forms of proof evidence are intended to have different implementations of proof certificates that are not necessarily known to the author of the LKF^a-based kernel. In practice, therefore, we avoid the anti-modular nature of this additional obligation by defining `copy` for certificate formats that are standard and well known. This suggests a two step process: first, we transform arbitrary proof certificates with Skolem terms into *max certs with Skolem terms* using the pairing technique of [7], which uses an unmodified LKF^a kernel and has a target certificate format that is common across all uses of the FPC checker. Then, we use a modified LKF^a kernel with the variant of the $\forall$ rule above to transform it into a *max cert with eigenvariables*, which would then be exported to the ultimate verifier written in Coq.

The adventurous Coqnoscenti might wonder whether it would be possible to repeat the λProlog-based certificate extraction outlined in Secs. 4 and 5 directly in Coq with a suitable library of Ltacs. We believe that this would be rather complicated because the logic programming strength of Ltacs is about that of ordinary Prolog, i.e., reasoning with Horn clauses. We are aware of current work on integrating the ELPI implementation of λProlog [16] into the Coq system in order to supply Coq with a powerful new extension language based on λProlog [35]. The entire framework proposed in this paper could perhaps be ported to Coq/ELPI, but that remains for future work.

7 Additional Observations

As we observed at the end of Section 4, the proof checking kernel described in this paper can handle outer skolemization well (at least in the case where the propositional connectives are polarized negatively). Unfortunately, pure outer skolemization can often insert Skolem functions with more arguments than are strictly necessary. Often automated theorem provers benefit from having Skolem functions with lower arity [31]. Thus, a natural question to ask is whether or not various methods used in practice for obtaining fewer arguments to Skolem functions can be certified.

7.1 Miniscoping and the Cut Rule

An important transformation technique on quantified formulas is *miniscoping*, which pushes quantifiers inwards as much as possible in order to minimize the scope of quantifiers. The *miniscoped* form of a formula is its normal form with respect to the rewrite system given by the following rules.

$$\begin{aligned}\forall x.\, (A \wedge B) &\longrightarrow (\forall x.\, A) \wedge (\forall x.\, B)\\ \exists x.\, (A \vee B) &\longrightarrow (\exists x.\, A) \vee (\exists x.\, B)\\ Qx.\, (A \star C) &\longrightarrow (Qx.\, A) \star C\\ Qx.\, (C \star A) &\longrightarrow C \star (Qx.\, A)\\ Qx.\, C &\longrightarrow C\end{aligned}$$

where $Q \in \{\forall, \exists\}$, $\star \in \{\wedge, \vee\}$, and x is not free in C. Miniscoping only involves changing the scopes of quantifiers, and does not otherwise change the logical structure of formulas: clearly the original and miniscoped formulas are logically equivalent. In particular, if $\tilde{B}$ is the miniscoped version of B, then checkable certificates of the sequent $\vdash \tilde{B}^{\perp}, B$ are easy to build. using a method for building proof certificates for term rewriting proof systems found in [10].

If we now skolemize $\tilde{B}$ and obtain proof evidence that is certifiable using the mechanisms described in this paper, then we have actually managed to get a (hybrid) proof certificate for the original formula B: simply use the cut inference rule in *LKF* and LKF^a to build a proof of $\vdash B$ from the proofs of $\vdash \tilde{B}^{\perp}, B$ and $\vdash \tilde{B}$. Note that we allow cut rules to be present within proof certificates and that "Skolem-elimination" does not imply "cut-elimination". If we were only interested in cut-free deskolemized proofs, then there can be a dramatic

increase in the size of a cut-free proof for $\vdash B$ given a cut-free proof of $\vdash \tilde{B}$ [5].

Optimization techniques for skolemization are often sophisticated: see, for example, [22] for a technique using BDDs that reduces dependencies on weak variables when performing skolemization. Any such optimization technique is compatible with our deskolemization procedure by means of cuts, just as with miniscoping, assuming an entailment between the optimized formulas and the original theorem can be proved and certified.

7.2 Skolemization and Polarities

When stating the conditions for the applicability of our deskolemization procedure at the end of Section 4, we have asked that the client use exclusively negative connectives, with the existential as the only positive connective. We can initially motivate this requirement with a consideration on the operational behavior of the procedure: positive connectives have the property that they force the proof checker to end a sequence of asynchronous rules, and possibly move the focus to a different subformula. A skolemized proof evidence could at this point use names for any eigenvariable. However, it might well be the case that the eigenvariable that corresponds to such a name has still not been instantiated, because it was to be created by an universal quantifier placed after the positive connective that caused the focus shift.

As a short example, consider the formula

$$((\forall x.\ \neg p(x)) \wedge \neg q) \vee \exists x.\, (p(x) \vee q).$$

Suppose we have proof evidence in the form of an *LK* proof for its skolemization $(\neg p(c) \wedge \neg q) \vee \exists x.\, (p(x) \vee q)$, with c a fresh Skolem constant. This means that we could be handed one of the following two proofs:

$$\dfrac{\dfrac{\dfrac{}{\vdash \neg p(c), p(c), q}\ \text{init} \quad \dfrac{}{\vdash \neg q, p(c), q}\ \text{init}}{\vdash \neg p(c) \wedge \neg q, p(c) \vee q}\ \wedge, \vee}{\vdash (\neg p(c) \wedge \neg q) \vee \exists x.\, (p(x) \vee q)}\ \vee, \exists(c)$$

$$\dfrac{\dfrac{\dfrac{}{\vdash \neg p(c), p(c), q}\ \text{init}}{\vdash \neg p(c), \exists x.\, (p(x) \vee q)}\ \exists(c), \vee \quad \dfrac{\dfrac{}{\vdash \neg q, p(c), q}\ \text{init}}{\vdash \neg q, \exists x.\, (p(x) \vee q)}\ \exists(c), \vee}{\vdash (\neg p(c) \wedge \neg q) \vee \exists x.\, (p(x) \vee q)}\ \vee, \wedge$$

Let's try to check the first proof against the unskolemized formula. The certificate will instruct the kernel to first apply the disjunction, and then instantiate the existential using the term c. The kernel will try to translate the client term c to a kernel term; however c is not in the signature, and there is no copy-clause generated by instantiating eigenvariables. Thus the check will fail! Indeed, this proof certificate also violates the precondition: if we polarize the skolemized formula negatively and try to check the *LK* proof against it, we can see that the kernel after applying the disjunction must proceed eagerly on the negative connectives and apply the negative conjunction. When instructed not to do so by the certificate, the check will fail. We can see that the negative polarization forces the proof to consume all the scope, and introduce all the needed eigenvariables, before proceeding with the existentials.

The choice of the example is not by chance: indeed, it comes directly from the example given in [4] for a class of formulas whose skolemizations can have proofs exponentially shorter than the originals. In order to be able to treat such cases, one would have to describe a potentially exponential deskolemization procedure; our wish is to maintain such a procedure to a lower complexity, hence our restriction to the case of negative connectives. Since we know that positive and negative connectives are equiprovable, it is then in general possible to accommodate more general proof evidences by making use of cuts.

7.3 The Topic of Inner Skolemization

Inner skolemization (see Definition 2.1) was introduced and proved sound by Andrews in [2]. His soundness proof fundamentally involved a model theoretic justification. As a result, we know of no systematic and proof theoretic means to certify proof evidence that results from using inner skolemization. However it is well known that deskolemization of inner skolemization is problematic [18]. The problem of inner skolemization turns out to be related to that of positive polarities: in either case, since we are able to suspend processing of the formula that would have yielded the eigenvariable in the corresponding unskolemized case, we get a "leakage" of variables (via their Skolem terms) from their scopes.

8 Related and Future Work

Summarizing, we have proposed an extension to the framework of Foundational Proof Certificates, that allows us to modularly extend definitions for various kinds of proof evidence in order to be able to check skolemized proofs. We have described the implementation of the improved kernel, and discussed some implemented examples.

There have been several different approaches to deskolemization in the past. Ours stands in contrast to approach of Reger and Suda [32] where certificates are allowed to involve inference rules that preserve satisfiability instead of provability; this was proposed there to treat, for example, skolemization. We shall not consider such extensions to the sequent calculus.

The problems discussed in Section 7 are well known in the literature. The example we used is a simplified form of the proof with exponential deskolemization from [4]; the deskolemization procedure described in that work for such cases is based on expansion trees, a formalism that is closely related to *LKF* proofs with negative connectives (see [8]). Färber and Kaliszyk [18] provide a deskolemization method that is related to our approach, and face similar problems. Their procedure also suffers from the problem with positively polarized connectives, although this is not described.

De Nivelle [15] performs deskolemization by introducing new predicate symbols that simulate Skolem functions. In contrast, we have tried to certify proofs by staying inside the original signature. The same author in [14] introduces reductions from various optimized skolemizations to a standard one in the spirit of our discussion at the beginning of Section 7; however that standard is inner skolemization, which is then certified by introducing a choice operator.

In the future we plan to study the interaction between positive polarities and skolemization and to extend this work to the higher-order setting starting from [26].

References

[1] Jean-Marc Andreoli. 1992. Logic Programming with Focusing Proofs in Linear Logic. *J. of Logic and Computation* 2, 3 (1992), 297–347. https://doi.org/10.1093/logcom/2.3.297

[2] Peter B. Andrews. 1981. Theorem Proving via General Matings. *J. ACM* 28, 2 (1981), 193–214. https://doi.org/10.1145/322248.322249

[3] Jeremy Avigad. 2003. Eliminating Definitions and Skolem Functions in First-Order Logic. *ACM Transactions on Computational Logic* 4 (2003), 402–415.

[4] Matthias Baaz, Stefan Hetzl, and Daniel Weller. 2012. On the complexity of proof deskolemization. *J. of Symbolic Logic* 77, 2 (2012), 669–686. https://doi.org/10.2178/jsl/1333566645

[5] Matthias Baaz and Alexander Leitsch. 1994. On Skolemization and Proof Complexity. *Fundamenta Informaticae* 20, 4 (1994), 353–379. https://doi.org/10.3233/FI-1994-2044

[6] Haniel Barbosa, Jasmin Christian Blanchette, and Pascal Fontaine. 2017. Scalable Fine-Grained Proofs for Formula Processing. In *26th International Conference on Automated Deduction (CADE) (LNCS)*, Leonardo de Moura (Ed.), Vol. 10395. Springer, 398–412. https://doi.org/10.1007/978-3-319-63046-5_25

[7] Roberto Blanco, Zakaria Chihani, and Dale Miller. 2017. Translating Between Implicit and Explicit Versions of Proof. In *Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings (Lecture Notes in Computer Science)*, Leonardo de Moura (Ed.), Vol. 10395. Springer, 255–273. https://doi.org/10.1007/978-3-319-63046-5_16

[8] Kaustuv Chaudhuri, Stefan Hetzl, and Dale Miller. 2016. A Multi-Focused Proof System Isomorphic to Expansion Proofs. *J. of Logic and Computation* 26, 2 (2016), 577–603. https://doi.org/10.1093/logcom/exu030

[9] Zakaria Chihani, Tomer Libal, and Giselle Reis. 2015. The proof certifier Checkers. In *Proceedings of the 24th Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX) (LNCS)*, Hans De Nivelle (Ed.). Springer, 201–210.

[10] Zakaria Chihani and Dale Miller. 2016. Proof Certificates for Equality Reasoning. In *Post-proceedings of LSFA 2015: 10th Workshop on Logical and Semantic Frameworks, with Applications. Natal, Brazil. (ENTCS)*, Mario Benevides and René Thiemann (Eds.). Elsevier, 93–108. https://doi.org/10.1016/j.entcs.2016.06.007

[11] Zakaria Chihani, Dale Miller, and Fabien Renaud. 2013. Checking Foundational Proof Certificates for First-Order Logic (extended abstract). In *Third International Workshop on Proof Exchange for Theorem Proving (PxTP 2013) (EPiC Series)*, J. C. Blanchette and J. Urban (Eds.), Vol. 14. EasyChair, 58–66.

[12] Zakaria Chihani, Dale Miller, and Fabien Renaud. 2017. A semantic framework for proof evidence. *J. of Automated Reasoning* 59 (2017), 287–330. https://doi.org/10.1007/s10817-016-9380-6

[13] Alonzo Church. 1940. A Formulation of the Simple Theory of Types. *J. of Symbolic Logic* 5 (1940), 56–68. https://doi.org/10.2307/2266170

[14] Hans de Nivelle. 2002. Extraction of Proofs from the Clausal Normal Form Transformation. In *CSL: 16th Workshop on Computer Science Logic (LNCS)*, Vol. 2471. LNCS, Springer-Verlag, 584–598.

[15] Hans de Nivelle. 2005. Translation of resolution proofs into short first-order proofs without choice axioms. *Information and Computation* 199, 1-2 (2005), 24–54.

[16] Cvetan Dunchev, Ferruccio Guidi, Claudio Sacerdoti Coen, and Enrico Tassi. 2015. ELPI: Fast, Embeddable, λProlog Interpreter. In *Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings (LNCS)*, Martin Davis, Ansgar Fehnker, Annabelle McIver, and Andrei Voronkov (Eds.), Vol. 9450. Springer, 460–468. https://doi.org/10.1007/978-3-662-48899-7_32

[17] Gabriel Ebner, Stefan Hetzl, Giselle Reis, Martin Riener, Simon Wolfsteiner, and Sebastian Zivota. 2016. System Description: GAPT 2.0. In *Proceedings of the 8th International Joint Conference on Automated Reasoning, IJCAR 2016 (LNCS)*, Nicola Olivetti and Ashish Tiwari (Eds.), Vol. 9706. Springer, 293–301. https://doi.org/10.1007/978-3-319-40229-1

[18] Michael Färber and Cezary Kaliszyk. 2016. No Choice: Reconstruction of First-order ATP Proofs without Skolem Functions. In *Proceedings of the 5th Workshop on Practical Aspects of Automated Reasoning (PAAR) (CEUR Workshop Proceedings)*, Pascal Fontaine, Stephan Schulz 0001, and Josef Urban (Eds.), Vol. 1635. CEUR-WS.org, 24–31.

[19] Pascal Fontaine, Jean-Yves Marion, Stephan Merz, Leonor Prensa Nieto, and Alwen Fernanto Tiu. 2006. Expressiveness + Automation + Soundness: Towards Combining SMT Solvers and Interactive Proof Assistants. In *TACAS: Tools and Algorithms for the Construction and Analysis of Systems (LNCS)*, H. Hermanns and J. Palsberg (Eds.), Vol. 3920. Springer, 167–181. https://doi.org/10.1007/11691372_11

[20] Gerhard Gentzen. 1935. Investigations into Logical Deduction. In *The Collected Papers of Gerhard Gentzen*, M. E. Szabo (Ed.). North-Holland, Amsterdam, 68–131. https://doi.org/10.1007/BF01201353

[21] Jean-Yves Girard. 1991. A new constructive logic: classical logic. *Math. Structures in Comp. Science* 1 (1991), 255–296. https://doi.org/10.1017/S0960129500001328

[22] Jean Goubault. 1995. A BDD-Based Simplification and Skolemization Procedure. *Logic Journal of the IGPL* 3, 6 (1995), 827–855.

[23] Ulrich Kohlenbach and Paulo Oliva. 2003. Proof mining in L_1-approximation. *Annals of Pure and Applied Logic* 121, 1 (2003), 1–38.

[24] Chuck Liang and Dale Miller. 2009. Focusing and Polarization in Linear, Intuitionistic, and Classical Logics. *Theoretical Computer Science* 410, 46 (2009), 4747–4768. https://doi.org/10.1016/j.tcs.2009.07.041

[25] Dale Miller. 1983. *Proofs in Higher-order Logic.* Ph.D. Dissertation. Carnegie-Mellon University. http://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/th.pdf

[26] Dale Miller. 1987. A Compact Representation of Proofs. *Studia Logica* 46, 4 (1987), 347–370.

[27] Dale Miller. 1990. Abstractions in logic programming. In *Logic and Computer Science*, Piergiorgio Odifreddi (Ed.). Academic Press, 329–359. http://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/AbsInLP.pdf.pdf

[28] Dale Miller. 1991. Unification of Simply Typed Lambda-Terms as Logic Programming. In *Eighth International Logic Programming Conference.* MIT Press, Paris, France, 255–269.

[29] Dale Miller and Gopalan Nadathur. 2012. *Programming with Higher-Order Logic.* Cambridge University Press. https://doi.org/10.1017/CBO9781139021326

[30] Gopalan Nadathur and Dustin J. Mitchell. 1999. System Description: Teyjus — A Compiler and Abstract Machine Based Implementation of λProlog. In *16th Conf. on Automated Deduction (CADE) (LNAI)*, H. Ganzinger (Ed.). Springer, Trento, 287–291.

[31] Andreas Nonnengart and Christoph Weidenbach. 2001. Computing Small Clause Normal Forms. In *Handbook of Automated Reasoning*,

Alan Robinson and Andrei Voronkov (Eds.). Vol. I. Elsevier Science B.V., Chapter 6, 335–367.

[32] Giles Reger and Martin Suda. 2017. Checkable Proofs for First-Order Theorem Proving. In *ARCADE 2017, 1st International Workshop on Automated Reasoning: Challenges, Applications, Directions, Exemplary Achievements (EPiC Series in Computing)*, Giles Reger and Dmitriy Traytel (Eds.), Vol. 51. EasyChair, 55–63. http://www.easychair.org/publications/paper/5W2B

[33] Joesph R. Shoenfield. 1967. *Mathematical Logic.* Addison-Wesley.

[34] Aaron Stump, Duckki Oe, Andrew Reynolds, Liana Hadarean, and Cesare Tinelli. 2013. SMT proof checking using a logical framework. *Formal Methods in System Design* 42, 1 (2013), 91–118.

[35] Enrico Tassi. 2018. Elpi: an extension language for Coq. CoqPL 2018: The Fourth International Workshop on Coq for Programming Languages. https://hal.inria.fr/hal-01637063/