A proof-theoretic approach to certifying skolemization

Kaustuv Chaudhuri, Matteo Manighetti, and Dale Miller

Inria & LIX, École Polytechnique
Palaiseau, France

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 also influenced by the polarities given to propositional connectives.

2012 ACM Subject Classification F.4.1. Mathematical logic: proof theory

Keywords and phrases proof certificates, skolemization, sequent calculus, focusing

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 resolution 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

---

\(^1\) 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.
A proof-theoretic approach to certifying skolemization

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 [19].

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 [12]. 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 $\epsilon$-operator and its associated axioms. Such operators can be used to specify Skolem functions; for instance, the $\epsilon$ 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 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 meta-theorems. 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, [18, 33] of proof evidence being imported into higher-order proof systems.) However, skolemization is not sound for higher-order logic (without choice) [25] and for intuitionistic logic, so the $LK$ proofs that can be imported need to be for the original unskolemized formulas.

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 [22]. 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 Our approach to deskolemization

Deskolemization has been widely studied for classical first-order logic. On the theoretical side various kinds of deskolemization results have been obtained for different forms of skolemization. For example, in [24, 25] 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.

In this paper we continue the study of checking and certifying proof evidence that contains
Skolem functions by explicitly deskolemizing proof evidence and building LK-style sequent
calculus proofs containing eigenvariables. Our approach to deskolemization can be described
as follows. We identify two different actors involved with proof checking. The client is some
theorem prover which wants to export checkable proofs and the kernel is a program that is
entrusted to check proofs in a completely trustworthy fashion. In this 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 [26]. Since it is impossible for a client to directly refer to
such abstractions, the client must make use of various naming mechanisms in order to refer
to those kernel-side abstractions. As we shall see, Skolem terms serve as one of these naming
mechanisms.

1.3 Summary of our contributions

This paper makes the following contributions to the problem of deskolemizing proof evidence.
1. We provide a modular method 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 structures [11]. 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 [28] and Elpi [15]).
3. We give a precise characterization of the surprising interaction of skolemization and
polarities arising from focused proofs. It turns out that positive polarities are just as
dangerous as inner skolemization, which is already well known to be difficult to deskolemize
syntactically [17, 4]. 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 \(s, t, \ldots\) will, as
usual, be built from variables \(x, y, \ldots\) and function applications of the form \(f(t_1, \ldots, t_n)\)
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 \((A, B, \ldots)\) and literals \((L)\) belong to the following grammar:

\[
A, B, \ldots ::= L \mid A \land B \mid \top \mid A \lor B \mid \bot \mid \forall x. A \mid \exists x. A \\
L ::= p \mid \neg p
\]

Here, \(p\) ranges over atomic formulas that are always of the form \(a(t_1, \ldots, t_n)\) 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 \(p/\neg p, \land/\lor, \top/\bot\) 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 \(\alpha\)-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 \(\Sigma_0\). In order to account for skolemization, we introduce another signature, \(\Sigma_{sk}\), disjoint with \(\Sigma_0\), 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 \(\Sigma_{sk}\) of that arity.

**Definition 1** (Skolemization). The following standard definitions are from [29].

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 \(x_1, \ldots, x_n (n \geq 0)\); and
- the second formula results from replacing that \(\forall x. C\) occurrence in \(B\) with the instance \([f(x_1, \ldots, x_n)/x]C\) where \(f\) is an \(n\)-arity symbol from \(\Sigma_{sk}\) that does not appear in \(B\).

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 \(f(y_1, \ldots, y_m)\) where \(y_1, \ldots, y_m\) 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 \(y_1, \ldots, y_m\) are contained in the list \(x_1, \ldots, x_n\).

**Example 2.** The Drinkers formula \(\exists x. \forall y. (\neg d(x) \lor d(y))\) can be skolemized as follows.

Outer: \(\exists x. (\neg d(x) \lor d(f(x)))\)

Inner: \(\exists x. (\neg d(x) \lor d(f())\)

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 doesn’t 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 [32, Section 4.5].
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 into the proof such as witness terms or choice paths, and making the wrong choice or guess 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 logic [1, 20] and then extended to a wide variety of classical, intuitionistic, and modal logics and other proof systems, focusing can be seen as a way of organizing proofs in such a way that choice points are minimized and the two types of choices are clearly separated. 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 $LKF$ [23, 11].

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 form the unpolarized case with the pairs $\land/\lor$, $\top/\bot$, and $\exists/\forall$.

\begin{align*}
A, B, \ldots &::= P | N & \text{(formulas)} \\
P, Q, \ldots &::= p | A \land B | \top | A \lor B | \bot | \exists x. A & \text{(positive formulas)} \\
N, M, \ldots &::= \neg p | A \land B | \top | A \lor B | \bot | \forall x. A & \text{(positive formulas)}
\end{align*}

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: $A \land B$ and $A \lor B$ are not only equi-provable, but each implies the other. However, positive and negative formulas have very different proofs, both in size and in shape.

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

\[ \Sigma \vdash \Gamma \not\vdash A, \Theta \quad \Sigma \vdash \Gamma \not\vdash B, \Theta \]
\[ \Sigma \vdash \Gamma \vdash \not\vdash, \Theta \quad \Sigma \vdash \Gamma \vdash \not\vdash B, \Theta \quad \Sigma \vdash \Gamma \vdash \not\vdash A \lor B, \Theta \quad \Sigma \vdash \Gamma \vdash \not\vdash A, \Theta \]
\[ \Sigma \vdash \Gamma \vdash \forall \cdot. A, \Theta \quad \Sigma \vdash \Gamma \vdash (\exists x. A) \]

Synchronous rules

\[ \Sigma \vdash \Gamma \vdash A \quad \Sigma \vdash \Gamma \vdash B \]
\[ \Sigma \vdash \Gamma \vdash A \land B \quad \Sigma \vdash \Gamma \vdash A \lor B \]
\[ \Sigma \vdash \Gamma \vdash \forall \cdot. A \quad \Sigma \vdash \Gamma \vdash \exists x. A \]

Identity rules

\[ \Sigma \vdash \Gamma, \neg p \vdash p \quad \text{init} \quad \Sigma \vdash \Gamma \vdash A \quad \Sigma \vdash \Gamma \vdash A^+ \quad \text{cut} \]

Structural rules

\[ \Sigma \vdash \Gamma, P \vdash P \quad \text{decide} \quad \Sigma \vdash \Gamma, R \vdash \Theta \quad \text{store} \quad \Sigma \vdash \Gamma \vdash N \quad \text{release} \]

In the store rule, \( R \) is a positive formula or a literal.

\( \square \) Figure 1 Rules of LKF. \( \Gamma \) is a multiset of positive formulas or literals, and \( \Theta \) is a list of formulas.

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 \( \land \) and \( \lor \) are the following (modulo certain minor differences):

\[ \vdash A, \Delta \quad \vdash B, \Delta \quad \vdash A \land B, \Delta \]
\[ \vdash A \lor B, \Delta \]

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 \( \lor \), for instance, the synchronous rules are:

\[ \vdash A, \Delta \quad \vdash B, \Delta \quad \vdash A \lor B, \Delta \]

These rules encode an essential choice between the two operands \( A \) and \( B \). The two polarized variants of \( \lor \) 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 means of two kinds of sequents:

\[ \Sigma \vdash \Gamma \vdash A \quad \text{synchronous sequent with } A \text{ under focus} \]
\[ \Sigma \vdash \Gamma \vdash \Theta \quad \text{asynchronous sequent} \]

The context \( \Gamma \), called the store, is a multiset of positive formulas or literals, and \( \Theta \), called the asynchronous zone, is a list of formulas. \( \Sigma \) 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 \( \Sigma \), written \( \Sigma \vdash (\text{wf } t) \) to mean that all the function symbols in \( t \) are used with the correct arities defined in \( \Sigma \), and that all the free variables of \( t \) are contained in the set of eigenvariables in \( \Sigma \).

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 and in the entire phase the focused formula is required to be principal. 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 neutral sequent, 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 [23].

\begin{theorem}[Soundness and Completeness] Let $B$ be a formula of first-order classical logic. If $B$ is a theorem, then $\vdash \cdot \uparrow \cdot \hat{B}$ is derivable for every polarized version $\hat{B}$ of $B$. Furthermore, if $\vdash \cdot \uparrow \cdot \hat{B}$ is provable for some polarized version $\hat{B}$ of $B$, then $B$ is a theorem. \end{theorem}

\section{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 [11]. 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 $\lor$-introduction or which witness term to use for $\exists$-introduction.

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

- $\Xi; \Sigma \vdash \Gamma \uparrow \cdot$ synchronous sequent with $A$ under focus
- $\Xi; \Sigma \vdash \Gamma \uparrow \cdot \Theta$ asynchronous sequent

Here, $\Xi$ 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 abstract.

\footnote{We will sometimes call such sequents neutral.}
Asynchronous rules

\[
\begin{align*}
\Xi_1; \Sigma \vdash \Gamma \uparrow A, \Theta & \quad \Xi_2; \Sigma \vdash \Gamma \uparrow B, \Theta \\
\Xi_0; \Sigma; \Gamma \vdash A \land B, \Theta & \quad \Xi_0; \Sigma; \Gamma \vdash \top, \Theta \\
\Xi_1; \Sigma; \Gamma \vdash A, B, \Theta & \quad \forall_c(\Xi_0, \Xi_1) \\
\Xi_0; \Sigma, \Gamma \vdash A \lor B, \Theta & \quad \Xi_1; \Sigma; \Gamma \vdash \bot, \Theta \\
\Xi_1; \Sigma, (\text{copy } t y) \vdash [y/x]A, \Theta & \quad \forall_c(\Xi_0, \Xi_1, t) \\
\Xi_0; \Sigma; \Gamma \vdash \forall x. A, \Theta & \quad y \notin \Sigma
\end{align*}
\]

Synchronous rules

\[
\begin{align*}
\Xi_1; \Sigma \vdash \Gamma \downarrow A & \quad \Xi_2; \Sigma \vdash \Gamma \downarrow B \\
\Xi_0; \Sigma; \Gamma \vdash A \land B & \quad \Xi_0; \Sigma; \Gamma \vdash \top \\
\Xi_1; \Sigma \vdash \Gamma \downarrow A_i & \quad \forall_c(\Xi_0, \Xi_1, i) \\
\Xi_0; \Sigma; \Gamma \vdash A_1 \lor A_2 & \quad i \in \{1, 2\} \\
\Xi_0; \Sigma; \Gamma \vdash \forall x. A & \quad \Xi_1; \Sigma \vdash \Gamma \downarrow (\text{copy } t s) \\
\Xi_0; \Sigma; \Gamma \vdash [s/x]A & \quad \exists_c(\Xi_0, \Xi_1, t)
\end{align*}
\]

Identity rules

\[
\begin{align*}
\text{init}_c(\Xi_0, l) & \quad \Xi_1; \Sigma; \Gamma \vdash l: \neg p \downarrow p \\
\text{cut}_c(\Xi_0, \Xi_1, \Xi_2, A) & \quad \Xi_0; \Sigma; \Gamma \vdash \top
\end{align*}
\]

Structural rules

\[
\begin{align*}
\Xi_1; \Sigma; \Gamma, l: P \vdash P & \quad \text{decide}_c(\Xi_0, \Xi_1, l) \\
\Xi_0; \Sigma; \Gamma; l: P \uparrow & \quad \Xi_1; \Sigma; \Gamma \vdash N & \quad \text{release}_c(\Xi_0, \Xi_1) \\
\Xi_0; \Sigma; \Gamma \vdash \top & \quad \text{store}_c(\Xi_0, \Xi_1, l) \\
\Xi_0; \Sigma; \Gamma \vdash R, \Theta & \quad \Xi_0; \Sigma; \Gamma \vdash \top
\end{align*}
\]

In the store rule, \(R\) is a positive formula or a literal.

\textbf{Figure 2} Rules of \(LKF^n\), an augmented version of \(LKF\). \(\Gamma\) is a multiset of pairs of the form \(l: R\) where \(l\) is an index and \(R\) is a positive formula or literal, and \(\Theta\) is a list of formulas.
Specifications and implementations of previous versions of proof checkers for the Foundational Proof Certificate framework [7, 10, 11] did not address the fact that client-side terms might be different than kernel-side terms. Since substitution terms are not always part of some particular presentation of proof evidence (since unification during proof checking can reconstruct such substitutions), 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 and we now describe how to extend the previous approach of FPC-based checkers to account for that difference.

The predicate \( \text{copy} \cdot \cdot \) 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 [27]. The copy-clauses based on the signature \( \{a/0, f/1, g/2\} \) have the following \( \lambda \text{Prolog} \) specification. (We do not assume any advance knowledge of \( \lambda \text{Prolog} \); for more information about that language, see [?].)

\[
\begin{align*}
\text{copy } a & \ a. \\
\text{copy } (f \ X) & \ (f \ U) :- \text{copy } X \ U. \\
\text{copy } (g \ X \ Y) & \ (g \ U \ V) :- \text{copy } X \ U, \text{copy } Y \ V.
\end{align*}
\]

It is easy to show that if \( t \) and \( s \) are two closed terms over the signature \( \{a/0, f/1, g/2\} \), then \( \text{(copy } t \ s) \) 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 \( \Sigma \) 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 \( \lambda \text{Prolog} \), as has been described in several other papers [7, 10, 11]. Although such implementations can be small, we present here only a few clauses. First, two simple clauses.

\[
\begin{align*}
\text{async Cert } ((A \ or- \ B)::R) & :- \text{orC Cert Cert } ', \text{async Cert } ' (A::B::R). \\
\text{sync Cert } (A \ or+ \ B) & :- \text{orE Cert Cert } C, \\
& (C = \text{left}, \text{sync Cert } ' A); \\
& (C = \text{right}, \text{sync Cert } ' B)).
\end{align*}
\]

Here, the proof theory judgments \( \Xi; \Sigma \vdash \Gamma \uparrow \Theta \) and \( \Xi; \Sigma \vdash \Gamma \downarrow A \) are represented by the atomic formulas \( \text{async Cert Theta} \) and \( \text{sync Cert A} \), respectively: the encoding of \( \Sigma \) and \( \Gamma \) are captured by features found in the (intuitionistic) logic underlying \( \lambda \text{Prolog} \). Thus, the two clauses above implement the intended meaning of the focused introduction rules for \( \lor \) and \( \land \), 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 \( \lambda \text{Prolog} \) clauses specifying the introduction of the quantifiers.

\[
\begin{align*}
\text{async Cert } ((\all B)::R) & :- \allCx Cert Cert ' T, \text{pi w\ copy T w } \Rightarrow \text{async Cert } ' (B \ w)::R). \\
\text{sync Cert } (\some B) & :- \someE Cert Cert ' T, \text{copy T E, sync Cert } ' (B \ E).
\end{align*}
\]

Note that the universal quantification of \( \lambda \text{Prolog} \ (\pi \ w \\backslash \text{copy } T \ w) \) implements the eigenvariable feature needed for the \( \text{LKF}^a \) proof system and that the implication \( \Rightarrow \) is used to extend the program clauses for \( \text{copy} \) with a new atomic clause \( \text{copy } T \ w \), which is only usable within the scope of \( w \). In this way, the \( \Sigma \) context in Figure 2 is implemented via \( \lambda \text{Prolog} \)'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 \( \Sigma_0 \) and that the copy-clauses \( C(\Sigma_0) \) derived from that signature are added to the kernel specification. As proof checking progresses, new atomic copy-formulas are added to the \( \Sigma \) context whenever a
strong quantifier is encountered (via the first clause displayed above). Whenever the client
computes (via the existential expert \texttt{someE}) a client-side term $T$ is then translated to the
kernel-side formula $S$ by the query $\text{copy } T S$.

\begin{example}
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 $\Gamma$ contains exactly the two
associations $(\text{copy } (h \ a) \ x)$ and $(\text{copy } (h \ (f \ a)) \ y)$. Attempting to prove the $\lambda$Prolog
query $C(\Sigma), \Gamma \vdash (\text{copy } (g \ (h \ (f \ a))) \ (f \ (h \ a))) \ X)$, 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 $\Gamma$ context contained the atoms $(\text{copy } (h \ a) \ x)$ and
$(\text{copy } (h \ a) \ y)$, then there are two solutions to the query $(\text{copy } (g \ (h \ a) \ (f \ a)) \ X)$,
namely, binding $X$ to either $(g \ x \ (f \ a))$ or $(g \ y \ (f \ a))$. \end{example}

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 $(\text{copy } t \ x)$ in the signature
with $(\text{wf } x)$, and dropping the indexes on the formulas in the store. Call this its \textit{underlying
sequent}. The following property is proved by a simple structural induction on $LKF^a$ proofs.

\begin{theorem}[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 $LK$. \end{theorem}

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 $\lambda$Prolog 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 [8, 9, 11]. 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. In a similar way, 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 $\land_e$, $\lor_e$, and $\top_e$
(since the positive propositional connectives do not appear) as well as the clerk $\forall_e$ (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.

\begin{verbatim}
allCx Cert Cert T.
\end{verbatim}
Given that these various assumptions hold, then we can prove the following: if it is checkable that the certificate $\Xi$ satisfies the FPC $\mathcal{P}$ as a proof of $C$ then the certificate $\Xi$ 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: https://github.com/chaudhuri/proofcert-deskolemize/. 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 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 [25] 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]. Also, the GAPT system [16] contains an implementation of that procedure.

5.1 Expansion trees with select variables

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

Definition 7 (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 $(e\text{Or } Q_1 Q_2)$ and $(e\text{And } Q_1 Q_2)$ are expansion trees for $A_1 \lor A_2$ and $A_1 \land A_2$ respectively.

If $u$ is a variable (called a select variable) and $Q$ is an expansion tree of $[u/x]A$, then $(e\text{All } u Q)$ is an expansion tree for $\forall x.A$.

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 $(e\text{Some } [(t_1, Q_1), \ldots, (t_n, Q_n)])$ is an expansion tree for $\exists x. A$.

Expansion terms can contain select variables, of course. 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
A proof-theoretic approach to certifying skolemization

kind et
type.

type eTrue, eFalse et.
type eLit et.
type eAnd, eOr 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 new primitive type et and the type declarations introduce new constructors for this primitive type.

kind address type.
type root address.
type lf, rg, dn address -> address.
type idx address -> index.
type abbrev context list (pair address et).
type astate, 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.

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 have made the assumption that only one term is present in the list: this is due to how contraction is treated which is done 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 5, 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 [25] 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 $\Sigma_0$.

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.

\[
\text{allCx Cert Cert T.}
\]

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 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 terms with a 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.
6.1 Miniscoping and the cut rule

An important transformation technique on quantified formulas is *miniscoping*, which consists in pushing 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.

\[
\begin{align*}
\forall x. (A \land B) & \rightarrow (\forall x. A) \land (\forall x. B) \\
\exists x. (A \lor B) & \rightarrow (\exists x. A) \lor (\exists x. B) \\
Q x. (A \ast B) & \rightarrow (Q x. A) \ast B \\
Q x. (B \ast A) & \rightarrow B \ast (Q x. A) \\
Q x. B & \rightarrow B
\end{align*}
\]

where \( Q \in \{\forall, \exists\} \) and \( \ast \in \{\land, \lor\} \). In the three rules that are marked by \( (\dagger) \), we assume that \( x \) is not free in \( B \). 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, it is an easy matter to prove that \( B \) entails \( \bar{B} \), where \( \bar{B} \) is the miniscoped version of \( B \). In fact, building a checkable proof certificate that the sequent \( \vdash \bar{B} \vdash B \) is a simple matter and could follow the method for building certificates for term rewriting proof systems [9].

If we now skolemize \( \bar{B} \) 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 \( LKFa \) to build a proof of \( \vdash B \) from the proofs of \( \vdash \bar{B} \vdash B \) and \( \vdash \bar{B} \). 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 \( \vdash \bar{B} \) [5].

Optimization techniques for skolemization can be rather sophisticated: see, for example, [21] 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.

6.2 Skolemization and polarities

When stating the conditions for the applicability of our deskolemization procedure, we have asked that the client use only negative connectives, with the existential as the only positive. 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 \((\forall x. \neg p(x)) \land \neg q) \lor \exists x. (p(x) \lor q)\). Suppose we have proof evidence in the form of an \( LK \) proof for its skolemization \((\neg p(c) \land \neg q) \lor \exists x. (p(x) \lor q)\), with \( c \) a fresh Skolem constant. This means that we could be handled one of the following two proofs:

\[
\begin{align*}
\vdash \neg p(c), p(c), q & \quad \text{init} \\
\vdash \neg q, p(c), q & \quad \text{init} \\
\vdash (\neg p(c) \land \neg q) \lor p(c) & \quad \lor, \land \\
\vdash \neg p(c) \lor q & \quad \lor
\end{align*}
\]

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.

6.3 The topic of inner skolemization

Inner skolemization (see Definition 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 [17]. The problem of inner skolemization turns out to be very similar 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.

7 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 the paper [30] by Reger and Suda, where certificates are allowed to involve inference rules that preserve satisfaction instead of provability: this was proposed there to treat, for example, skolemization. We shall not consider such extensions to proof certificates.

The problematics discussed in Section 6 are well known in the literature. The running example is a simplified form of the proof with exponential deskolemization from [4]; Färber and Kaliszyk [17] provide a method that can ultimately be seen as an instance of our approach, and show that there are problems with inner skolemization—we provided here a better explanation of this phenomenon. De Nivelle [14] 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 [13] introduces reductions from various optimized skolemizations to a standard one in the spirit of our discussion at the beginning of Section 6; however that standard is inner skolemization, which is then certified by introducing a choice operator.

In the future we wish to study more the interaction between positive polarities and skolemization. Other lines of work include extending this to the higher-order setting. Skolemization works similarly with higher-order quantification [25], and we expect our approach to naturally extend to this case.

References

A proof-theoretic approach to certifying skolemization


16 Gabriel Ebner, Stefan Hetzl, Giselle Reis, Martin Riener, Simon Wollweber, and Sebastian Zivota. System description: GAPT 2.0. In Nicola Olivetti and Ashish Tiwari, editors, *Pro-


A proof-theoretic approach to certifying skolemization