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 implementations of Gentzen's sequent calculus, which directly support 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 connects skolemized terms to the internal eigenvariable abstractions assumes that *outer* skolemization has been used. Many variations and optimizations of skolemization are certified by allowing formulas to be manipulated into equivalent forms (e.g., miniscoping) prior to applying outer skolemization: the resulting certified proof retains a cut in which such an equivalent formula is used as a lemma.

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

Keywords and phrases certification, skolemization, sequent calculus, focused proof system **FIXME**

1 Introduction

Skolemization is a process (of which there are many variants) that removes *strong quantifiers* by instantiating such quantifiers with terms generally of the form $(f\ x_1 \ldots\ x_n)$ where $n \geq 0$ and $x_1, \ldots, x_n$ is a list of distinct *weakly quantified variables*.[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. Those implementing 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. Cut-free sequent calculus proofs of skolemized formulas do not contain occurrences of eigenvariables.

The correctness of skolemization in first-order classical logic is generally justified by referring to the model theory of classical logic. In particular, the main meta-theorem surrounding 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 proof) then 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

[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.

classical logic again, we know that B is valid and, hence, by completeness we know that B has a proof. A central issue surrounding the use of skolemization is how to actually *export* the proof (or refutation) of a skolemized version of $\neg B$ so that we can formally *certify* that B is a theorem.

In this paper we are interested in *certification* in the sense of having proofs formally checked using computerized proof-checkers. One method to formally certify a proof using skolemization, is to formally prove the model-theoretic completeness of first-order classical logic in a formal reasoning system such as Coq or Isabelle/HOL. In order to complete such a proof, significant aspects of the foundations of mathematics would be employed, including axioms of extensionality, infinity, and choice [10]. Certifying that B is provable would then require two steps: (1) formally checking the proof evidence supplied that skolemized version of $\neg B$ is unsatisfiable (by checking, for example, that a refutation refutation is syntactically correct) and then (2) applying the formalized metatheory of classical logic following the outline offered above.

A theorem prover containing a choice operator, such as Hilbert's ϵ-operator and its associated axioms, can potentially provide a more targeted justification for using Skolem functions since such functions can be specified using choice operators. Such a use of the ϵ operator for justifying skolemization has been used in Isabelle/HOL [6]. However, this still leaves unsolved the problem of a direct certification of a skolemized proof in a system with weaker foundations (without the axiom of choice, say) and therefore lacks such operators.

Another more elementary and direct approach would be to *deskolemize* the proof into a proof in, say, Gentzen's sequent calculus LK, which is complete for classical first-order logic without relying on choice operators or axioms. One does not then need any of the powerful proof techniques behind completeness and choice principles. Instead, one only needs to check that a proposed proof structure does, indeed, describe a formal sequent calculus proof.

1.1 Advantages of building sequent calculus proofs

While both certification using formalized model theory or using choice operators are sufficient to convince most people that theorems proved using Skolem terms are, in fact, true (and therefore provable), there are a number of reasons why it is important to push harder to actually build proofs without Skolem functions.

For example, skolemization is not, in general, sound for higher-order logic (without choice) [20] and for intuitionistic logic. If it is possible to build sequent style proofs without Skolem functions, it should be possible to import such proofs directly into higher-order provers. It might also be possible to judge that the resulting sequent calculus proof is intuitionistically valid or not, thereby allowing it to be imported into provers based only on intuitionistic provers (see, for example, [15, 28] of proof evidence being imported into higher-order proof systems).

It is also the case that if we can provide Gentzen-style LK-proof from a proof that used skolemized proof evidence, we have a secured a low-level logic proof of the theorem for which we can have a high-degree of confidence. Such a proof should be importable into a wider range of provers, in particular, provers that do not assume choice principles.

We can imagine future work that involves interacting with, browsing, and mining [18] formal proof structures. If that proof relies on just, say, Gentzen's LK, then the resulting interactions should be rather direct and informative. If that proof relies on mathematical results about choice principles preserving satisfiability with logics that are extended with Skolem functions, then that interaction is likely to be 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 [23, 20] 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 more commonly 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 Gentzen's *LK*-style sequent calculus proofs containing eigenvariables. As we shall see, our approach to deskolemization can be viewed as in a programming language setting in two ways. First, we identify two different *actors* of a proof checker. The *client* is some theorem prover who 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 [21]. 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 into the construction of a sequent calculus proof in classical first-order logic. Modularity is explicitly provided by the structure of the kernel used in the Foundational Proof Certificate (FPC) framework for defining proof structures [9]. This proof checking framework builds Gentzen-style *LK* sequent calculus proofs using eigenvariables: such sequent proofs are essentially performed and are not generally stored. For example, outer skolemization proof evidence (without the use of cuts) leads to cut-free and Skolem-free *LK* proofs.
2. Prior to performing skolemization, provers often move quantifiers within a formula (e.g, anti-prenexing) in order to reduce the number of arguments need when building Skolem terms. By shortening the list of arguments to Skolem functions, theorem provers can expect to find shorter proofs. We provide a simple mechanism that allows the checking of proofs that employ such optimizations: such optimizations are encoded using a cut (i.e., a lemma).
3. 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 every successful rule of 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: in this way, one does not need to trust the particular implementations of λProlog we have used (Teyjus [24] and Elpi [12]).

KC patrolled here on 2018-04-16 15:56:34+0200

2 Skolemization

As is customary, we shall assume that all formulas are in *negation formal form*: that is, negations have only atomic scope and the only logical connectives are $\wedge$, $\vee$, $\top$, $\bot$, $\forall$, and $\exists$. 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 shall also assume that no two occurrences of a quantifier (either $\forall$ or $\exists$) bind variables with the same name. Alphabetic change of bound variables always make this possible.

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.

Function symbols come equip with arity a collection of function symbols with their arity is called a *signature*. An example of a signature is $\{a/0, f/1, g/2\}$. We also assume that the set of terms generated from a signature is non-empty (for example, the collection $\{f/1, g/2\}$ is not a signature) and that a symbol is given at most one arity within a signature.

We shall assume that all first-order formulas for which we perform proof checking contain function symbols and constants from the fix signature Σ_0. In order to account for skolemization, we introduce another signature, Σ_{sk}, 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.

The following definition, which we take from [25], seems to be standard. An *outer skolemization step* is a pair of formulas in which

1. the first formula, say, B is such that if contains the subformula $\forall x.C(x)$ that is not in the scope of any universal quantifier and which is in the scope of existential quantifiers binding the variables $x_1, \ldots, x_n$ $(n \geq 0)$.
2. the second formula results from replacing that $\forall x.C(x)$ occurrence in B with $C(f(x_1, \ldots, x_n))$ where f is an n-arity symbol from Σ_{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 $C(x)$ is $f(y_1, \ldots, y_m)$ where $y_1, \ldots, y_m$ are the free variables of the occurrence of $\forall x.C(x)$. Notice that necessarily, $m \leq n$ and that all the variables in the list $y_1, \ldots, y_m$ are contained in the list $x_1, \ldots, x_n$.

The formula E is the result of performing *outer skolemization* on B if there is a sequent of *outer skolemization step* 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 sequent of *outer skolemization step* that carries B to E and where E does not contain any strong quantifiers (i.e., universal quantifiers).

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 [**?**, Section 4.5].

▶ **Theorem 1.** *Let B be a first-order formula over the signature Σ_0 and let E is either an inner or outer skolemization of B. If $\neg B$ is satisfiable then $\neg E$ is satisfiable.*

3 Focused Sequent calculus

In Section 1.1, we argued that building explicit sequent calculus proofs can benefit the certification process. Although we wish to build (or at least perform) a sequent calculus proof in the sense of Gentzen [16], the construction of such proofs can be highly chaotic. The certification process can be viewed as a kind of protocol between two agents. One agent is client whose has already found proof evidence, say, a resolution refutation or an expansion proof. The other agent is the kernel which contains a highly trusted implementation of, say, Gentzen's *LK* sequent calculus proof system. It is the desire of the client to send instructions to the kernel in order to guide the kernel to build a complete sequent proof. Note that the kernel does not have to build and store the resulting sequent calculus proof: it will be enough that the kernel preforms it.

Given this description of the certification process, we can see that directly employing the original sequent calculus could be highly problematic. Attempting to build a proof of a sequent can, in principle, depend a great deal of communication to go between the client and the kernel since nearly every sequent can be the conclusion of a structure rules (weakening and contraction), a cut rule, and a (possibly large) number of inference rules. And once the client instructs the kernel to attempt one such inference rule, it is likely that one or two new sequents (the premises of the applied rule) need proofs and that each of these requires again essentially the same kind of instructions. Clearly, such a simplistic kernel, with its demand to organized it "micro-rules", puts an enormous burden on the client.

Fortunately, recent advances in the sequent calculus, namely, the discovery of polarization and focusing—first developed for linear logic [1, 17] and then extended to classical and intuitionistic logic—have made it possible to design highly structured and greatly reduced protocols between such clients and kernels. For example, as we shall see, logical connectives with *negative* polarity have invertible introduction rules: those when these connectives appear in a sequent, no communication needs to take place between the client and kernel since the kernel can simply eagerly apply the inference rules associated to invertible connectives without loss of provability. Similarly, connectives with *positive* connectives are selected as a *focus* of the kernel's proof building process: the kernel may need to ask the client to suggest non-invertible rules to apply in this case, but as long as the focus remains, the kernel can be structured to only request help with that one formula (and not the many other formulas surrounding it in the sequent). In this section, we details of a focused proof system for first-order classical logic and in the next section, we describe how to formally exploit such highly structured sequent calculus proofs to yield the protocol between client and kernel that is the basis of the *foundational proof certificate* framework.

The basis of our certification is a variant of the *LKF* proof system [19], which is a sequent calculus for classical first-order logic given in the Gentzen-Schütte style (a.k.a. Tait style), based on the system GS[1, 2, 3] [29]. *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 arity n. Formulas $(A, B, \dots)$ will belong to the following grammar, which we divide into the two *polarities*, *positive* $(P, Q, \dots)$ and *negative* $(N, M, \dots)$, that we explain below.

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

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

$$N, M, \dots ::= \neg\, p \mid A \mathbin{\overset{-}{\wedge}} B \mid \overset{-}{\top} \mid A \mathbin{\overset{-}{\vee}} B \mid \overset{-}{\bot} \mid \forall x.\, A \qquad \text{(positive formulas)}$$

$$L ::= p \mid \neg\, p \qquad \text{(literals)}$$

Here, p ranges over positive *atomic formulas* that are always of the form $a(t_1, \dots, t_n)$ where a is some predicate symbol of arity n. We write $A^\perp$ for the de Morgan dual of A, given by the pairs $p / \neg\, p$, $\overset{+}{\wedge}/\overset{-}{\vee}$, $\overset{+}{\top}/\overset{-}{\bot}$, $\overset{+}{\vee}/\overset{-}{\wedge}$, $\overset{+}{\bot}/\overset{-}{\top}$, and $\exists/\forall$.

For the non-quantifier connectives, the polarity amounts to an annotation on the formula; the quantifiers, on the other hand, have a unique polarity. The polarity annotations do not

Asynchronous rules

$$\frac{\Sigma \vdash \Gamma \Uparrow A, \Theta \quad \Sigma \vdash \Gamma \Uparrow B, \Theta}{\Sigma \vdash \Gamma \Uparrow A \wedge^- B, \Theta} \qquad \frac{}{\Sigma \vdash \Gamma \Uparrow t^-, \Theta} \qquad \frac{\Sigma \vdash \Gamma \Uparrow A, B, \Theta}{\Sigma \vdash \Gamma \Uparrow A \vee^- B, \Theta} \qquad \frac{\Sigma \vdash \Gamma \Uparrow \Theta}{\Sigma \vdash \Gamma \Uparrow f^-, \Theta} \qquad \frac{\Sigma, y \vdash \Gamma \Uparrow [y/x]A, \Theta}{\Sigma \vdash \Gamma \Uparrow \forall x.\, A, \Theta}\; y \notin \Sigma$$

Synchronous rules

$$\frac{\Sigma \vdash \Gamma \Downarrow A \quad \Sigma \vdash \Gamma \Downarrow B}{\Sigma \vdash \Gamma \Downarrow A \wedge^+ B} \qquad \frac{}{\Sigma \vdash \Gamma \Downarrow t^+} \qquad \frac{\Sigma \vdash \Gamma \Downarrow A}{\Sigma \vdash \Gamma \Downarrow A \vee^+ B} \qquad \frac{\Sigma \vdash \Gamma \Downarrow B}{\Sigma \vdash \Gamma \Downarrow A \vee^+ B} \qquad \frac{\Sigma \vdash (\mathtt{wf}\ t) \quad \Sigma \vdash \Gamma \Downarrow [t/x]A}{\Sigma \vdash \Gamma \Downarrow \exists x.\, A}$$

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

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

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.

affect the truth of a formula, so $A \wedge^+ B$ and $A \wedge^- B$ are equivalent. However, positive and negative formulas have very different proofs. Intuitively, the introduction rules for negative formula are *invertible*: that is, these rules have the property that their collection of premises are *equivalent* to the conclusion. These invertible inference rules are organized into the *asynchronous phase*: that is, a grouping of inference rules for which the kernel can apply without needing to communicate with the client. For instance, the rules for $\wedge^-$ and $\vee^-$ are the following (modulo certain minor differences explained below):

$$\frac{\vdash A, \Delta \quad \vdash B, \Delta}{\vdash A \wedge^- B, \Delta} \qquad \frac{\vdash A, B, \Delta}{\vdash A \vee^- B, \Delta}$$

A positive (non-atomic) formula, on the other hand, has an inference rules that is not necessarily invertible, meaning that its introduction rule may involve a choice and its premise(s) may not be equivalent to its conclusion. As a result, such inference rules are organized into the *synchronous phase*: that is, a grouping of inference rules for which the kernel needs to communicate with the client. For $\vee^+$, for instance, the synchronous rules are:

$$\frac{\vdash A, \Delta}{\vdash A \vee^+ B, \Delta} \qquad \frac{\vdash B, \Delta}{\vdash A \vee^+ B, \Delta}$$

These rules encode an essential choice between the two operands A and B. The benefit of having both polarized variants of $\vee$ is that our framework will be able to build more proofs in a more flexible fashion.

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

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

where 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 is a set of *eigenvariables* that can be free in the terms to the right of $\vdash$. An asynchronous sequent of the form $\Sigma \vdash \Gamma \Uparrow \cdot$ is called a *neutral sequent*.

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 neutral sequent as conclusion; a positive formula is chosen for *focus* and in the entire phase the focused formula is required to be principal. The synchronous phase may close the proof with the init rule when the focused formula is an atom, 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 *failure* since there is no other inference rule available to prove a focus on a positive literal.) 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 propositional formula. We shall also assume that atomic formulas are polarized arbitrarily: they could be all negative, all positive, or some mixture of these two. The following theorem is proved in [19].

▶ **Theorem 2** (Soundness and Completeness of *LKF*)**.** *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.*

Note that polarization does not affect provability but it can and does have significant impact on the size and shape of proofs.

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 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 [9]. The key idea is to augment the *LKF* proof system as follows.

- A *proof certificate* is threaded througth every sequent and inference rule: these certificates are term structures that contain the clients proof evidence.
- Additional premises are added to the *LKF* inference rules: these premises manipulate and extract information from proof certificates and serve as guards or handlers for inference rules.

There are two kinds of additional premises added to inference rules. The first kind, the *clerks*, are added to asynchronous rules: clerks perform routine maintance 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 $\vee$-introduction or which witness term to use for $\exists$-introduction.

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

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

Both of the structures Σ and Γ are generalized in the LKF^a over what they were in *LKF*. In particular, Σ is now more than a signature: is a set of pairing of the form $(\mathtt{copy}\ t\ y)$

Asynchronous rules

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

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

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

Synchronous rules

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

$$\frac{\Xi_1; \Sigma \vdash \Gamma \Downarrow A_i \quad \vee_e(\Xi_0, \Xi_1, i)}{\Xi_0; \Sigma \vdash \Gamma \Downarrow A_1 \mathbin{\overset{+}{\vee}} A_2}\ i \in \{1, 2\} \qquad \frac{\Sigma \vdash (\mathtt{copy}\ t\ s) \quad \Xi_1; \Sigma \vdash \Gamma \Downarrow [s/x]A \quad \exists_e(\Xi_0, \Xi_1, t)}{\Xi_0; \Sigma \vdash \Gamma \Downarrow \exists x.\, A}$$

Identity rules

$$\frac{\mathrm{init}_e(\Xi_0, l)}{\Xi_0; \Sigma \vdash \Gamma, l{:}\neg p \Downarrow p}\ \mathrm{init} \qquad \frac{\Xi_1; \Sigma \vdash \Gamma \Uparrow A \quad \Xi_2; \Sigma \vdash \Gamma \Uparrow A^{\perp} \quad \mathrm{cut}_e(\Xi_0, \Xi_1, \Xi_2, A)}{\Xi_0; \Sigma \vdash \Gamma \Uparrow \cdot}\ \mathrm{cut}$$

Structural rules

$$\frac{\Xi_1; \Sigma \vdash \Gamma, l{:}P \Downarrow P \quad \mathrm{decide}_e(\Xi_0, \Xi_1, l)}{\Xi_0; \Sigma \vdash \Gamma, l{:}P \Uparrow \cdot}\ \mathrm{decide} \qquad \frac{\Xi_1; \Sigma \vdash \Gamma \Uparrow N \quad \mathrm{release}_e(\Xi_0, \Xi_1)}{\Xi_0; \Sigma \vdash \Gamma \Downarrow N}\ \mathrm{release}$$

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

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 formulas or literals, and Θ is a list of formulas.

where t is a client-side term (containing, for example, Skolem functions) 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 pair of the form $l{:}R$ where l is an *index* and R is a positive formula or a literal. The exact structures behind indexes is not specified by the kernel but is 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 $\overset{-}{\wedge}_c$ since clerks are defined 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). There are also additional arguments that may be indexes (in the case of init_e, decide_e, and store_c), 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).

The predicate (`copy · ·`) 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 [22]. The copy-clauses based on the signature $\{a/0, f/1, g/2\}$ have the following λProlog specification.

```
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 be two closed terms over the signature $\{a/0, f/1, g/2\}$, `copy t s` is provable from these clauses if and only if $t = s$. Obviously, an arbitrary first-order signature can be translated into such a set of copy-clauses.

The inference rules in Figure 2 can be implemented directly in λProlog, as has been described in several other papers [7, 8, 9]. Although such implemementations can be small, we present here only a few clauses. First, two simple clauses.

```
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 proof theory judgments $\Xi; \Sigma \vdash \Gamma \Uparrow \Theta$ 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 the of focused introduction rules for $\bar{\vee}$ and $\dot{\vee}$, 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.[2]

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

Note that the universal implication of λProlog (`pi w\`) implements the eigenvariable feature needed for the LKF^a proof system and that the implication `=>` is used to assume the atomic fact `(copy T 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 both agree on the signature Σ_0 and that the copy-clauses derived from that signature, say, $\mathcal{C}(\Sigma_0)$. As proof checking progresses, new copy-atomic formulas are added to the Σ context whenever a strong quantifier is encounter (via the first clause displayed above). Whenever the client computes (via the existential expert `someE`) a client-side term `T` is then translated to the kernel-side formula `S` by the query `copy T S`.

▶ **Example 3.** Assume that the base signature for both the client and the kernel is $\{a/0, f/1, g/2\}$. 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 the associations

[2] The explicit translation based on copy-clauses was not a feature of earlier FPC kernels [7, 8, 9]: in those earlier paper, substitution terms were either not stored in proof certificates or there were no difference between client-side and kernel-side terms since theorem did not contain strong quantifiers. Maybe this should be said more plainly since otherwise reviewers might think that this section has nothing new in it.

`(copy (h a) x)` and `(copy (h (f a)) y)`. Then the λProlog query `(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 determiniate. 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 example 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 and the indexes on the formulas in the store; call this its *underlying sequent*. The following property is obvious.

▶ **Theorem 4** (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 LK provable.*

The completeness of LKF^a depends on the specification of the clerk and expert predicates supplied by the client. 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. This property is a critical feature of a proof checking kernel.

DM Still to consider to some extent:

- Describe FPCs
- Talk about client vs. kernel views (indexes, polarities)
- We still need to be clear that polarization maps formulas and skolemization maps terms from client to kernel.
- There is also the parallel between skolem-terms-as-names and indexes-as-surrogate formulas.

5 Various forms of skolemization

DM will continue here by about 21:00.

Goal here: return to the topic of outer skolemization and inner too. What is a general approach to justifying these different approaches? Use a cut-formula, etc.

NOTE (Matteo): I am not sure whether it is actually useful to have a paragraph dedicated to miniscoping, given that the treatment is analogous to any other optimization. The question about inner skolemization still prevents me from having a very clear view of this section.

Different kinds of skolemization (e.g., outer vs inner) can greatly influence the complexity of proof search. For this reason, theorem provers employ various kinds of optimizations to the standard skolemization in order to have more control on the term generation. The simplest of these techniques consists in moving, when possible, quantifiers in or out over other connectives. In some cases, proofs will contain smaller Skolem terms while in other cases proofs will contain fewer but bigger terms. Optimizations techniques for skolemization can be rather sophisticated: see, for example, [?] for a technique using BDDs that reduces dependencies on weak variables when performing skolemization. In this section we will see how we can justify proof evidence obtained with uses of some optimizations.

5.1 Miniscoping

Very often automated theorem provers benefit from having Skolem terms with a lower arity [27]. The most important transformation technique to this aim is *Miniscoping*, consisting in pushing quantifiers as inwards as possible, in order to minimize the scope of quantifiers. Let's formally define what a miniscoped formula is.

▶ **Definition 5** (Miniscoping rules, miniscoped formula).

- $\forall x\ (\varphi(x,z) \wedge \psi(x,z)) \mapsto (\forall x_1\ \varphi(x_1,z)) \wedge (\forall x_2\ \psi(x_2,z))$
- $\exists x\ (\varphi(x,z) \vee \psi(x,z)) \mapsto (\exists x_1\ \varphi(x_1,z)) \vee (\exists x_2\ \psi(x_2,z))$
- $\mathcal{Q}x\ (\varphi(x,z) \circ \psi(z)) \mapsto (\mathcal{Q}x\ \varphi(x,z)) \circ \psi(z)$
- $\mathcal{Q}x\ (\varphi(z) \circ \psi(x,z)) \mapsto \varphi(z) \circ (\mathcal{Q}x\ \psi(x,z))$

Where $\mathcal{Q}$ is any of $\forall, \exists$ and $\circ$ is any of $\wedge, \vee$. A formula is said to be *miniscoped* if none of the miniscoping rules is applicable.

Miniscoping only involves changing the scope of quantifiers, and doesn't otherwise change the logical structure of formulas: clearly the original and miniscoped formulas are logically equivalent. It is however very well known [5] that this can have a dramatic impact on the size of cut-free deskolemized proofs.

We take a different approach here and allow the cut rule in the checking procedure. We also require the client to communicate that miniscoping has been applied prior to skolemization.

Given a formula F, it is then easy to compute its miniscoped version F', and then produce proof evidence that $F' \vdash F$. We can then check the skolemized proof evidence against the unskolemized F' with our technique, and a single cut will yield proof evidence for F.

5.2 Other optimizations

NOTE (Matteo): De Nivelle [11] is a good citation for transforming various optimized skolemizations to standard ones. I didn't include it since it targets inner skolemization, and our stance on this is unclear.

The discussion about miniscoping can be generalized to other kind of optimization. A theorem prover could apply any number of clever operations to a formula when it believes that it will obtain better results when applying skolemization.

We require for these cases that the client describes these operations. In the case of miniscoping, the entailment could be easily checked; in the case of more clever optimizations, the client will also need to provide justifications for them.

The optimizations will finally be included in the proof checking procedure in the form of cuts, as it was the case for miniscoping.

5.3 The topic of inner skolemization

Thus, a cut-free proof using outer-skolemization yields a cut-free proof using inner-skolemization. The converse is, however, not necessarily true.

We might be faced with a situation in which we have a cut-free proof of $sk_i(B)$ (using inner skolemization) but no simple way to construct a cut-free proof of B. This is a topic (really? check this) addressed by Baaz and others. Since inner skolemization is sound (proved by Andrews?), then the existence of a proof of inner skolemization of B means that B is valid and, hence, it has a cut-free proof (by completeness and cut-elimination). The resulting proof size can be much larger (again, Baaz et al).

Relate innermost skolemization with miniscoping. Matteo has a counterexample to the claim: innermost skolemization is the same as miniscoping and then using outermost skolemization.

Thus, we might need to resign to using "miniscoping plus outermost" as opposed to innermost. If we live with this limitation, then we can automatically generate the cut/lemma formula. (The generation of the formal proof of entailment with miniscoped formulas is still a bit tricky...)

6 Experiments with an implementation

Several experimental implementations were carried out, adapting existing FPC code and crafting new ones to demonstrate the improved kernel. They are available on the web at *(ref)*.

Here we will concentrate on describing the implementations concerning Expansion Trees. Expansion Trees [20] are a proof formalism that generalizes the idea of Herbrand disjunctions to formulas with arbitrary quantifiers. We assume here formulas to be in negation normal form, and polarize negatively all the connectives. We can define the Expansion Tree of a formula F in the following way:

▶ **Definition 6** (Expansion Tree)**.**
- If A is an atomic formula, it is an Expansion Tree for itself
- If Q_1, Q_2 are Expansion Trees of F_1, F_2, then $\mathit{eOr}\, Q1\, Q2$ and $\mathit{eAnd}\, Q1\, Q2$ are Expansion Trees for $F_1 \vee F_2$ and $F_1 \wedge F_2$ respectively
- If u is a variable (called *select variable*) and Q is an expansion tree of F, then $\mathit{eAll}\, u\, Q$ is an Expansion Tree for $\forall x\, F$
- If $t_1, \ldots, t_n$ is a list of terms and Q is an Expansion Tree for F, then $\mathit{eSome}\, [(t_1, Q), \ldots, (t_n, Q)]$ is an Expansion Tree for $\exists x\, F$

Terms in existential nodes can make use of select variables. The original definition states in addition to this some correctness criteria: we do not need them in this context, since correctness is guaranteed by the kernel.

Expansion trees are also central in the deskolemization procedure described in [4], of which implementations exist (such as GAPT, [13]). Indeed, the notions of select variables and of terms using them puts Expansion Trees very close to the realm of skolemization: select variables can be seen as nothing but another mechanism for naming eigenvariables, in the spirit of client vs. kernel terms.

We implemented three procedures for checking different kinds of proof evidence based on this formalism: one for Expansion Trees, one for the slightly different Skolem Expansion Trees, and one for Expansion Trees of skolemized formulas.

6.1 Expansion Trees

We start from describing a basic FPC that can check proof evidence in the form of an expansion tree. The signature of the FPC, described in figure 3, contains two certificate constructors: `astate` is consumed during the asyncronous phase and records two contexts representing the storage and the asynchronous zone; `sstate` is consumed during the synchronous phase and records the storage and the formula under focus. Formulas are paired in the certificate with the expansion trees to which they are associated. The only index constructor uses the formulas themselves as indexes.

```
kind et         type.  % expansion trees
kind qet        type.  % quantified trees (leading introduction of select
    vars)

type idx              form -> index.

typeabbrev subExp     list (pair i et).     % Expansions for a node
typeabbrev context    list (pair form et). % Basic elements of contexts

type eIntro             (i -> qet) -> qet.
type eC                 et -> qet.

type eLit,eTrue, eFalse      et.
type eAnd, eOr          et -> et -> et.
type eAll               i  -> et -> et.
type eSome              subExp    -> et.

type astate             context -> context         -> cert.
type sstate             context -> (pair form et) -> cert.
```

Figure 3 Certificate constructors for Expansion Trees

```
orC     (astate Left ((pr B (eOr E1 E2))::Qs))
        (astate Left ((pr B1 E1)::(pr B2 E2)::Qs)) :- disj- B1 B2 B.
andC    (astate Left ((pr B (eAnd E1 E2))::Qs))
        (astate Left ((pr B1 E1)::Qs))
        (astate Left ((pr B2 E2)::Qs))             :- conj- B1 B2 B.
allCx   (astate Left ((pr ForallB (eAll Uvar E))::Qs))
        (w\ astate Left ((pr (B w) E)::Qs)) Uvar :- all- B ForallB.
someE   (sstate Left (pr Form (eSome [pr Term ExTree])))
        (astate Left [(pr (Body Term) ExTree)])
        Term :- some+ Body Form.
```

Figure 4 FPC for expansion trees

The main clerks and experts are presented in figure 4. Since connectives are polarized negatively, most of the work is carried out by clerks that simply consume the connectives and trees and add the components to the updated state.

When meeting a strong quantifier, the expansion tree contains the select variable associated to it: we will then use the new `allCx` to instruct the kernel to create a new eigenvariable, and provide the select variable as client name for that eigenvariable.

When we meet an existential node, together with the list of terms by which the existential should be instantiated, we can simply communicate the client term T to the kernel, that will proceed to translate it to a kernel term. Note that in the code we made the assumption that only one term is present in the list: this is due to how contratction is treated, and is outside of the current scope.

6.2 Skolem Expansion Trees

Skolem Expansion Trees are a structure introduced in the usual process of deskolemization through expansion trees. The only difference with usual expansion trees is that universal nodes are not instantiated by select variables, but by Skolem terms. Thus we can see them as a kind of skolemized proof evidence, where the client preserves the information of the association between Skolem terms and eigenvariables.

Given the discussion on the FPC for Expansion Tree, it is clear that this new setting does not provide any challenge to the old FPC: it is just presented with slightly different looking client terms, and will work exactly in the same way.

6.3 Expansion Trees of Skolemized formulas

We now turn our attention to a fully skolemized setting, where we are given an expansion tree relative to a skolemized formula and we want to check it against the original one. We only need to apply to the FPC the modification that was introduced in Section ??. First, we note that since there are no strong quantifiers left in the skolemized formula, the Expansion Tree will not contain any select variable node. Accordingly, we modify the ALLCX clerk in the following way

```
allCx   (astate Left ((pr ForallB E)::Qs))
        (w\ astate Left ((pr (B w) E)::Qs)) Sk :- all- B ForallB.
```

Thus, when the checker finds a strong quantifier it will be instructed to create a new eigenvariable, and it will use a logic variable as the name for it. This variable will ultimately be unified with the correct Skolem term.

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.

Future lines of work include extending this to the higher-order setting. Skolemization work similarly at higher-order quantification, and we expect our approach to naturally extend to this case. There have been several different approaches to deskolemization in the past. Ours stands in contrast to the paper [26] by Reger and Suda, where certificates are allowed to involve inference rules that preserve satisfaction: this was proposed there to treat, for example, Skolemization. We shall not consider such extensions to proof certificates here.

Färber and Kaliszyk [14] proceed in a similar way as we did, but obtain less general result as the work is directly linked to the formalisms of Resolution and Natural Deduction. De Nivelle[?] proceeds to deskolemization introducing new predicate symbols that simulate Skolem functions. This is in contrast with our spirit of staying inside the original signature.

References

1 Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *J. of Logic and Computation*, 2(3):297–347, 1992. doi:10.1093/logcom/2.3.297.

2 Peter B. Andrews. Theorem proving via general matings. *J. ACM*, 28(2):193–214, 1981. doi:10.1145/322248.322249.

3 Jeremy Avigad. Eliminating definitions and skolem functions in first-order logic. *ACM Transactions on Computational Logic*, 4:402–415, 2003.

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

5 Matthias Baaz and Alexander Leitsch. On skolemization and proof complexity. *Fundamenta Informaticae*, 20(4):353–379, 1994. URL: https://content.iospress.com/articles/fundamenta-informaticae/fi20-4-4, doi:10.3233/FI-1994-2044.

6 Haniel Barbosa, Jasmin Christian Blanchette, and Pascal Fontaine. Scalable fine-grained proofs for formula processing. In Leonardo de Moura, editor, *26th International Conference on Automated Deduction (CADE)*, volume 10395 of *LNCS*, pages 398–412. Springer, 2017. URL: https://doi.org/10.1007/978-3-319-63046-5_25, doi:10.1007/978-3-319-63046-5_25.

7 Roberto Blanco, Zakaria Chihani, and Dale Miller. Translating between implicit and explicit versions of proof. In Leonardo de Moura, editor, *Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *Lecture Notes in Computer Science*, pages 255–273. Springer, 2017. URL: https://doi.org/10.1007/978-3-319-63046-5_16, doi:10.1007/978-3-319-63046-5_16.

8 Zakaria Chihani, Dale Miller, and Fabien Renaud. Checking foundational proof certificates for first-order logic (extended abstract). In J. C. Blanchette and J. Urban, editors, *Third International Workshop on Proof Exchange for Theorem Proving (PxTP 2013)*, volume 14 of *EPiC Series*, pages 58–66. EasyChair, 2013.

9 Zakaria Chihani, Dale Miller, and Fabien Renaud. A semantic framework for proof evidence. *J. of Automated Reasoning*, 59:287–330, 2017. doi:10.1007/s10817-016-9380-6. URL: http://dx.doi.org/10.1007/s10817-016-9380-6, doi:10.1007/s10817-016-9380-6.

10 Alonzo Church. A formulation of the Simple Theory of Types. *J. of Symbolic Logic*, 5:56–68, 1940.

11 Hans de Nivelle. Extraction of proofs from the clausal normal form transformation. In *CSL: 16th Workshop on Computer Science Logic*, volume 2471 of *LNCS*, pages 584–598. LNCS, Springer-Verlag, 2002.

12 Cvetan Dunchev, Ferruccio Guidi, Claudio Sacerdoti Coen, and Enrico Tassi. ELPI: fast, embeddable, λProlog interpreter. In Martin Davis, Ansgar Fehnker, Annabelle McIver, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings*, volume 9450 of *LNCS*, pages 460–468. Springer, 2015. URL: http://dx.doi.org/10.1007/978-3-662-48899-7_32, doi:10.1007/978-3-662-48899-7_32.

13 Gabriel Ebner, Stefan Hetzl, Giselle Reis, Martin Riener, Simon Wolfsteiner, and Sebastian Zivota. System description: GAPT 2.0. In Nicola Olivetti and Ashish Tiwari, editors, *Proceedings of the 8th International Joint Conference on Automated Reasoning, IJCAR 2016*, volume 9706 of *LNCS*, pages 293–301. Springer, 2016. doi:10.1007/978-3-319-40229-1.

14 Michael Färber and Cezary Kaliszyk. No choice: Reconstruction of first-order ATP proofs without skolem functions. In Pascal Fontaine, Stephan Schulz 0001, and Josef Urban, editors, *Proceedings of the 5th Workshop on Practical Aspects of Automated Reasoning (PAAR)*, volume 1635 of *CEUR Workshop Proceedings*, pages 24–31. CEUR-WS.org, 2016.

15 Pascal Fontaine, Jean-Yves Marion, Stephan Merz, Leonor Prensa Nieto, and Alwen Fernanto Tiu. Expressiveness + automation + soundness: Towards combining SMT solvers and interactive proof assistants. In Holger Hermanns and Jens Palsberg, editors, *TACAS: Tools and Algorithms for the Construction and Analysis of Systems, 12th International Conference*, volume 3920 of *LNCS*, pages 167–181. Springer, 2006. `doi:10.1007/11691372\_11`.

16 Gerhard Gentzen. Investigations into logical deduction. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131. North-Holland, Amsterdam, 1935. `doi:10.1007/BF01201353`.

17 Jean-Yves Girard. A new constructive logic: classical logic. *Math. Structures in Comp. Science*, 1:255–296, 1991. `doi:10.1017/S0960129500001328`.

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

19 Chuck Liang and Dale Miller. Focusing and polarization in linear, intuitionistic, and classical logics. *Theoretical Computer Science*, 410(46):4747–4768, 2009. `doi:10.1016/j.tcs.2009.07.041`.

20 Dale Miller. A compact representation of proofs. *Studia Logica*, 46(4):347–370, 1987.

21 Dale Miller. Abstractions in logic programming. In Piergiorgio Odifreddi, editor, *Logic and Computer Science*, pages 329–359. Academic Press, 1990. URL: `http://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/AbsInLP.pdf.pdf`.

22 Dale Miller. Unification of simply typed lambda-terms as logic programming. In *Eighth International Logic Programming Conference*, pages 255–269, Paris, France, June 1991. MIT Press.

23 Dale A. Miller. *Proofs in Higher-order Logic*. PhD thesis, Carnegie-Mellon University, August 1983. URL: `http://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/th.pdf`.

24 Gopalan Nadathur and Dustin J. Mitchell. System description: Teyjus — A compiler and abstract machine based implementation of λProlog. In H. Ganzinger, editor, *16th Conf. on Automated Deduction (CADE)*, number 1632 in LNAI, pages 287–291, Trento, 1999. Springer.

25 Andreas Nonnengart and Christoph Weidenbach. Computing small clause normal forms. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 6, pages 335–367. Elsevier Science B.V., 2001.

26 Giles Reger and Martin Suda. Checkable proofs for first-order theorem proving. In Giles Reger and Dmitriy Traytel, editors, *ARCADE 2017, 1st International Workshop on Automated Reasoning: Challenges, Applications, Directions, Exemplary Achievements*, volume 51 of *EPiC Series in Computing*, pages 55–63. EasyChair, 2017. URL: `http://www.easychair.org/publications/paper/5W2B`.

27 J. Alan Robinson and Andrei Voronkov. *Handbook of Automated Reasoning*. MIT Press, 2001.

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

29 A. S. Troelstra and H. Schwichtenberg. *Basic Proof Theory*. Cambridge University Press, 2 edition, 2000.