A proposal for broad spectrum proof certificates

Dale Miller

INRIA & LIX,École Polytechnique

Abstract. Recent developments in the theory of *focused proof systems* provide flexible means for structuring proofs within the sequent calculus. This structuring is organized around the construction of "macro" level inference rules based on the "micro" inference rules which introduce single logical connectives. After presenting focused proof systems for first-order classical logics (one with and one without fixed points and equality) we illustrate several examples of proof certificates formats that are derived naturally from the structure of such focused proof systems. In principle, a proof certificate contains two parts: the first part describes how macro rules are defined in terms of micro rules and the second part describes a particular proof object using the macro rules. The first part, which is based on the vocabulary of focused proof systems, describes a collection of macro rules that can be used to directly present the structure of proof evidence captured by a particular class of computational logic systems. While such proof certificates can capture a wide variety of proof structures, a proof checker can remain simple since it must only understand the micro-rules and the discipline of focusing. Since proofs and proof certificates are often likely to be large, there must be some flexibility in allowing proof certificates to elide subproofs: as a result, proof checkers will necessarily be required to perform (bounded) proof search in order to reconstruct missing subproofs. Thus, proof checkers will need to do unification and restricted backtracking search.

1 Introduction

Most computational logic systems work in isolation in the sense that they are unable to communicate to each other documents that encode formal proofs that they can check and trust. We propose a framework for designing such documents which we will call *proof certificates*. Being based on foundational aspects of proof theory, these proof certificates hold the promise of working as a common communication medium for a broad spectrum of computational logic systems. Since formal proofs are often large, there can be significant time and space costs in producing, communicating, and checking proof certificate. A central aspect of the framework described here is that it provides for flexible trade-offs between these computational resources. In particular, proof certificates can be made smaller by removing subproofs: in that case, the proof checker must do (bounded) proof search to reconstruct elided proofs.

After presenting the basics of both sequent calculus and focused sequent calculus for first-order classical logic, we present several examples of proof certificates including those based on matrix and non-matrix (e.g., resolution) formats.

We then strengthen first-order logic to include both fixed points and equality: these extensions allow focused proof systems to immediately capture (non-deterministic) computation as well as some model-checking primitives. Additional proof certificates are then possible with these extensions to logic. We then conclude with a brief discussion of related and future work.

2 Proof theory and proof certificates

We shall assume that the reader has some familiarity with the sequent calculus. Here we recall some basic definitions and concepts.

2.1 The basics of sequent calculus

Sequents are a pair $\Gamma \vdash \Delta$ of two (possibly empty) collections of formulas. For Gentzen, these collections were lists but we shall assume that these collections are multisets. Such sequents are also called *two-sided* sequents: formulas on the left-hand-side (in Γ) are viewed as assumptions and formulas on the right-hand-side (in Δ) are viewed as possible conclusions: thus, an informal reading of the judgment described by the sequent $\Gamma \vdash \Delta$ is "if all the formulas in Γ are true then some formula in Δ is true."

Sequent calculus proof systems for classical, intuitionistic, and linear logics come with inference rules in which a sequent is the conclusion and zero or more sequents are premises. We break these rules down into three classes of rules. There are two kinds of *structural rules*:

$$\text{Contraction: } \frac{\Gamma, B, B \vdash \Delta}{\Gamma, B \vdash \Delta} \quad \frac{\Gamma \vdash \Delta, B, B}{\Gamma \vdash \Delta, B} \qquad \text{Weakening: } \frac{\Gamma \vdash \Delta}{\Gamma, B \vdash \Delta} \quad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, B}$$

The *identity rules* are also just two.

$$\frac{}{B \vdash B}\ \mathit{Initial} \qquad \frac{\Gamma \vdash \Delta, B \qquad \Gamma', B \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\ \mathit{Cut}$$

The meta-theory of most sequent calculus presentations of logic includes results that say that most instances of these identity rules are, in fact, not necessary. The cut-elimination theorem states that removing the cut-rule does not change the set of provable sequents. Furthermore, the initial rule can usually be eliminated for all cases except when B is an *atomic* formula, that is, a formula in which the top-level constant is a non-logical (predicate) symbol.

The third and final collection of inference rules are the *introduction rules* which describe the role of the logical connectives in proof. In two-sided sequent calculus proofs, these are usually organized as right and left introduction rules for the same connective. For example, here are two pairs of introduction rules for two connectives.

$$\frac{\Gamma, B, B' \vdash \Delta}{\Gamma, B \wedge B' \vdash \Delta} \qquad \frac{\Gamma \vdash \Delta, B \qquad \Gamma' \vdash \Delta', B'}{\Gamma, \Gamma' \vdash \Delta, \Delta', B \wedge B'} \qquad \frac{\Gamma, B[t/x] \vdash \Delta}{\Gamma, \forall x B \vdash \Delta} \qquad \frac{\Gamma \vdash \Delta, B[y/x]}{\Gamma \vdash \Delta, \forall x B}$$

The right-introduction rule for $\forall$ has the proviso that the *eigenvariable* y does not have a free occurrence in any formula in the sequent in the conclusion of the rule. Notice that in both of these sets of rules, there is exactly one new occurrence of a logical connective in the conclusion when compared to the premise(s). Some introduction rules are invertible: that is, if their conclusion is provable then all their premises are provable. Of the four introduction rules above, the left introduction rule for $\wedge$ and the right introduction rule for $\forall$ are invertible: the other two introduction rules are not necessarily invertible.

When presenting a sequent calculus proof system for a specific logic, one usually presents the introduction rules for the logical connectives of the logic and usually accepts both identity inference rules (initial and cut). The structural rules are, however, seldom adopted without restriction. For example, intuitionistic logic is a two-sided sequent calculus in which contraction on the right is not allowed. Multiplicative-additive linear logic (MALL) admits neither weakening nor contraction and full linear logic allows those structural rules only for specially marked formulas (formulas marked with the so-called *exponentials* ! and ?). Classical logic, however, generally admits these structural rules unrestricted.

2.2 Encoding computation with the sequent calculus

The sequent calculus can be used to encode computation as follows. A sequent, as a collection of formulas, can be used to encode the state of a computation. As one attempts to build a (cut-free) proof of a given initial sequent, new sequents appear. The dynamics of computation can be encoded by the changes that take place as one moves from conclusion sequent to premise sequents. The cut rule and the cut-elimination theorem are generally used not as part of computation but as a means to reason about computation. This *proof search* approach to specification has been used to formalize the operational semantics of logic programming [16]. Notice that this approach to encoding computation differs significantly from the approach inspired by the "Curry-Howard correspondence" in which natural deduction proofs can be seen as (functional) programs, normalization represent the process of computing, and normal forms represent values [14]. As is argued later in this paper and in [15], the proof search approach to specifying computation is natural for capturing proof checking and proof reconstruction.

If we try to take the construction of proofs literally as a model for performing computation, one is immediately struck by the inappropriateness of sequent calculus for this task: there are just too many ways to build proofs and most of them differ in truly inconsequential ways. While permuting the application of inference rules may yield proofs of different sequents, quite often such permutations yield different proofs of the same sequent. One would wish to have a much tighter correspondence between the application of an inference rule and something that might appear as an interesting "action" within a computation or "high-level" inference rule. Such a correspondence is possible but it requires adding more structure to the sequent calculus.

3 Focused proof systems

A normal form of sequent calculus that was designed to link steps in computation with steps in deduction can be found in the work on *uniform proofs* and *backchaining* [16] that was used to provide a proof theoretic foundation for logic programming. Andreoli generalized that work to provide a *focused proof system* [1] that allows one to richly restrict and organize the sequent calculus for linear logic. We provide here a high-level outline of the key ideas behind focused proof systems in the context of classical logic.

Focused proofs are divided into two, alternating phases. The first phase groups together all invertible inference rules. The second phase starts by selecting a formula on which to "focus": the inference rule that is applied to this formula is not necessarily invertible. Furthermore, the (reverse) application of that introduction rule will generate one or more (premise) sequents containing zero or more subformulas of the focus formula. If any of those subformulas require a non-invertible inference rule, then this phase continues with these subformulas as the new foci. This second phase, also called the *positive* phase, ends when either the proof ends with an instance of the initial rule or when the focus becomes one needing an invertible inference rule. Certain "structural" rules are used to recognize the end of a phase or the switch from one phase to another.

3.1 LKF: A focused proof system for classical logic

To illustrate these general comments about focused proof systems more concretely, we now present the LKF proof system for first-order classical logic [12]. We shall adopt a presentation of first-order classical logic in which negations are applied only to atomic formulas (*i.e.*, negation normal form) and where the propositional connectives t, f, $\wedge$, and $\vee$ are replaced by two "polarized" versions: t^-, t^+, f^-, f^+, $\wedge^-$, $\wedge^+$, $\vee^-$, $\vee^+$. Additionally, we assume that the atomic formulas are assigned positive or negative polarity following some arbitrary and fixed rule. A formula is *negative* if it is a negative atom, the negation of a positive atom, or if its top-level connective is one of t^-, f^-, $\wedge^-$, $\vee^-$, $\forall$. A formula is *positive* if it is a positive atom, the negation of a negative atom, or if its top-level connective is one of t^+, f^+, $\wedge^+$, $\vee^+$, $\exists$. Notice that taking the De Morgan dual of a formula causes its polarity to flip. Finally, a formula is a *literal* if it is an atom or a negated atom.

The LKF focused proof system for classical logic is given in Figure 1. Since we now restrict our attention to classical logic, we can simplify sequents by making them *one-sided*: that is, we can write the sequent $\Gamma \vdash \Delta$ as $\vdash \neg\Gamma, \Delta$ (placing $\neg$ in front of a collection of formulas is taken as the collection of negation normal negated formulas). In this setting, right and left-introduction rules are now organized around two right introduction rules for a connective and its De Morgan dual. Sequents in LKF are divided into *negative sequents* $\vdash \Theta \Uparrow \Gamma$ and *positive sequents* $\vdash \Theta \Downarrow B$, where Θ and Γ are multisets of formulas and B is a formula. (These sequents are formally one-sided sequents: formulas on the left of $\Uparrow$ and $\Downarrow$ are *not* negated as they are in two-sided sequents.) Notice that in this

Structural Rules

$$\frac{\vdash \Theta, C \Uparrow \Gamma}{\vdash \Theta \Uparrow \Gamma, C}\ \mathit{Store} \qquad \frac{\vdash \Theta \Uparrow N}{\vdash \Theta \Downarrow N}\ \mathit{Release} \qquad \frac{\vdash P, \Theta \Downarrow P}{\vdash P, \Theta \Uparrow \cdot}\ \mathit{Decide}$$

Identity Rules

$$\frac{\vdash \Theta \Downarrow P \quad \vdash \Theta \Uparrow \neg P}{\vdash \Theta \Uparrow}\ \mathit{Cut} \qquad \frac{}{\vdash \neg P, \Theta \Downarrow P}\ \mathit{Init}\ (\text{literal } P)$$

Introduction of negative connectives

$$\frac{}{\vdash \Theta \Uparrow \Gamma, t^-} \qquad \frac{\vdash \Theta \Uparrow \Gamma, A \quad \vdash \Theta \Uparrow \Gamma, B}{\vdash \Theta \Uparrow \Gamma, A \wedge^- B}$$

$$\frac{\vdash \Theta \Uparrow \Gamma}{\vdash \Theta \Uparrow \Gamma, f^-} \qquad \frac{\vdash \Theta \Uparrow \Gamma, A, B}{\vdash \Theta \Uparrow \Gamma, A \vee^- B} \qquad \frac{\vdash \Theta \Uparrow \Gamma, A}{\vdash \Theta \Uparrow \Gamma, \forall x A}$$

Introduction of positive connectives

$$\frac{}{\vdash \Theta \Downarrow t^+} \qquad \frac{\vdash \Theta \Downarrow A \quad \vdash \Theta \Downarrow B}{\vdash \Theta \Downarrow A \wedge^+ B}$$

$$\frac{\vdash \Theta \Downarrow A_1}{\vdash \Theta \Downarrow A_1 \vee^+ A_2} \qquad \frac{\vdash \Theta \Downarrow A_2}{\vdash \Theta \Downarrow A_1 \vee^+ A_2} \qquad \frac{\vdash \Theta \Downarrow A[t/x]}{\vdash \Theta \Downarrow \exists x A}$$

Fig. 1. The focused proof system LKF for classical logic. Here, P is positive, N is negative, C is a positive formula or a negative literal, Θ consists of positive formulas and negative literals, and x is not free in Θ and Γ. Endsequents have the form $\vdash \cdot \Uparrow \Gamma$.

focused proof system, we have reused the term "structural rule" for a different set of rules. The weakening and contraction rules are each available in exactly one rule in Figure 1, namely, in the *Init* and the *Decide* rules, respectively. Notice also that in any LKF proof that has a conclusion of the form $\vdash \cdot \Uparrow B$, the only formulas occurring to the left of an $\Uparrow$ or $\Downarrow$ within sequents in that proof are positive formulas or negative literals. There are three immediate consequences of this invariant. (*i*) The proviso on the *Init* rule (that P is a literal) is necessarily satisfied. (*ii*) The only formulas that are weakened (in the *Init* rule) are either positive formulas or negative literals. (*iii*) The only formulas contracted (in the *Decide* rule) are positive formulas. Although linear logic is not employed here directly, non-literal negative formulas are treated linearly in the sense that they are never duplicated nor weakened in an LKF proof.

Let B be a formula of first-order logic. By a *polarization* of B we mean a formula, say B', where all the propositional connectives are replaced by *polarized versions* of the same connective and where all atomic formulas are assigned either a positive or negative polarity. Thus, an occurrence of the disjunction $\vee$ is replaced by an occurrence of either $\vee^+$ or $\vee^-$; similarly with $\wedge$ and with the logical constants for true t and false f. For simplicity, we shall assume that polarization for atomic formulas is a global assignment to all atomic formulas. Properly speaking, focused proof systems contain *polarized* formulas and not

simply formulas. Notice that if the formula has n occurrences of these four logical connectives then there are 2^n different polarizations of that formula. The following theorem is proved in [12].

Theorem 1 (Soundness and completeness of LKF). *Let B be a first order formula and let B' be a polarization of B. Then B is provable in classical logic if and only if there is a cut-free LKF proof of $\vdash \cdot \Uparrow B'$.*

Notice that polarization does not affect provability but it does affect the shape of possible LKF proofs. To illustrate an application of the correctness of LKF, we show how it provides a direct proof of the following theorem.

Theorem 2 (Herbrand's Theorem). *Let B be a quantifier-free formula and let $\bar{x}$ be a (non-empty) list of variables containing the free variables of B. The formula $\exists \bar{x} B$ is classically provable if and only if there is a list of substitutions $\theta_1, \ldots, \theta_m$ ($m \geq 1$), all with domain $\bar{x}$, such that the (quantifier-free) disjunction $B\theta_1 \vee \cdots \vee B\theta_m$ is provable (*i.e.*, tautologous).*

Proof. Assume that $\exists \bar{x} B$ is provable and let B' be the result of polarizing all occurrences of propositional connectives negatively. By the completeness of LKF, there is a cut-free LKF proof Ξ of $\vdash \exists \bar{x} B' \Uparrow \cdot$. The only sequents of the form $\vdash \Theta \Uparrow \cdot$ in Ξ are such that Θ is equal to $\{\exists \bar{x} B'\} \cup \mathcal{L}$ for $\mathcal{L}$ a multiset of literals. Such a sequent can only be proved by a *Decide* rule that focuses on either a positive literal in $\mathcal{L}$ (in which case the proof is completed by the *Init* rule) or the original formula $\exists \bar{x} B'$: in the latter case, the positive phase above it provides a substitution for all the variables in $\bar{x}$. One only needs to collect all of these substitutions into a list $\theta_1, \ldots, \theta_m$ and then show that the proof Ξ is essentially also a proof of $\vdash B'\theta_1 \vee^+ \cdots \vee^+ B'\theta_m \Uparrow \cdot$ in the sense that the positive and negative phases correspond exactly. □

3.2 Positive and negative macro inference rules

We shall call individual introduction rules (such as displayed in Figure 1) "micro-rules" (the atoms of inference). An entire phase within a focused proof can be seen as a "macro-rule" (the molecules of inference). In particular, consider the following derivation, where P is a positive formula in Θ.

$$\dfrac{\dfrac{\vdash \Theta \Uparrow N_1 \quad \cdots \quad \vdash \Theta \Uparrow N_n}{\vdash \Theta \Downarrow P}}{\vdash \Theta \Uparrow \cdot}$$

Here, the selection of the formula P for the focus can be seen as selecting among several macro-rules: this derivation illustrates one such macro-rule: the inference rule with conclusion $\vdash \Theta \Uparrow \cdot$ and with $n \geq 0$ premises $\vdash \Theta \Uparrow N_1, \ldots, \vdash \Theta \Uparrow N_n$ (where $N_1, \ldots, N_n$ are negative formulas). We shall say that this macro-rule is positive. Similarly, there is a corresponding negative macro-rule with conclusion, say, $\vdash \Theta \Uparrow N_i$, and with $m \geq 0$ premises of the form $\vdash \Theta, \mathcal{C} \Uparrow \cdot$, where $\mathcal{C}$ is a multiset of positive formulas or negative literals.

In this way, focused proofs allow us to view the construction of proofs from conclusions of the form $\vdash \Theta \Uparrow \cdot$ as first attaching a positive macro rule (by focusing on some formula in Θ) and then attaching negative inference rules to the resulting premises until one is again to sequents of the form $\vdash \Theta' \Uparrow \cdot$. Focused proofs are built by such alternation of positive and negative macro-rules.

Example 3. Assume that Θ contains the formula $a \wedge^+ b \wedge^+ \neg c$, where a, b, and c are positive atomic formulas. A derivation that focuses on that formula must have the following shape.

$$\dfrac{\dfrac{}{\vdash \Theta \Downarrow a}\ \mathit{Init} \quad \dfrac{}{\vdash \Theta \Downarrow b}\ \mathit{Init} \quad \dfrac{\dfrac{\vdash \Theta, \neg c \Uparrow \cdot}{\vdash \Theta \Uparrow \neg c}\ \mathit{Store}}{\vdash \Theta \Downarrow \neg c}\ \mathit{Release}}{\dfrac{\vdash \Theta \Downarrow a \wedge^+ b \wedge^+ \neg c}{\vdash \Theta \Uparrow \cdot}\ \mathit{Decide}}$$

This derivation is possible only if Θ is of the form $\neg a, \neg b, \Theta'$. Thus, the corresponding "macro-rule" is

$$\dfrac{\vdash \neg a, \neg b, \neg c, \Theta' \Uparrow \cdot}{\vdash \neg a, \neg b, \Theta' \Uparrow \cdot}\ .$$

Thus, selecting this formula corresponds to the "action" of adding the literal $\neg c$ to the context if the two literals $\neg a$ and $\neg b$ are already present.

The *decide depth* of an LKF proof is the maximum number of *Decide* rules along any path starting from the endsequent. We shall often use the decide depth of proofs to help judge their size: as we shall see, such a measurement is more natural than the measurement that counts occurrences of micro rules.

4 Some examples of proof certificates

Let B be a classical propositional formula in negation normal form. Thus, every connective in B can be given either positive or negative polarity. We now consider the two extremes in which all the connectives are made negative and in which all the connectives are made positive.

Roughly speaking, we shall view proof certificates as documents containing two parts. The first part, the *preamble*, uses the language of focusing (*e.g.*, polarization of connectives and literals) to define macro-level connectives. The second part, the *payload*, contains the direct encoding of a proof using those macro-level connectives.

Example 4. Let B^- be the result of polarizing negatively the connectives of B: that is, B^- contains only the connectives $\wedge^-$, $\vee^-$, t^-, and f^-. In this case, an LKF proof of $\vdash \Uparrow B^-$ has a simple structure: in fact, it has a decide depth of exactly 1. The unique negative phase comprises *all* the introduction rules,

leaving one premise for every disjunct in the conjunctive normal form of B. Let one such premise be $\vdash L_1, \ldots, L_j \Uparrow$, where $L_1, \ldots, L_j$ are literals. Such a sequent is provable if and only if it has an LKF proof of the form

$$\dfrac{\dfrac{}{\vdash L_1, \ldots, L_j \Downarrow P}\ \mathit{Init}}{\vdash L_1, \ldots, L_j \Uparrow}\ \mathit{Decide}$$

where P is a positive literal from the set $\{L_1, \ldots, L_j\}$ and the complement of P is also in that set. Thus a proof certificate for propositional logic can be described as follows. The preamble declares that all propositional connectives are polarized negatively and that all atoms are polarized, say, negatively. Given this preamble, a proof checker will be able to compute the unique negative macro rule. The only information that is missing from the proof is the actual "mating" of complementary literals [2]. Thus, the payload needs to contain one pair of *occurrences* of literals for each disjunct in the conjunctive normal form of B: the first is positive and is used in the *Decide* rule and the second is the complement of the first and provides the information needed for the Init rule.

The proof certificate described in Example 4 is potentially large since it must provide a pair of literal occurrences for every one of the (exponentially many) clauses within the formula B. If we allow proof checkers to also do some simple "proof search" then this certificate can be made to have *constant* size. In this case, the proof certificate can simply tell the proof checker that it should search for a proof of *decide depth* 1 for every premise of the negative phase. In this setting, such proof search is trivial. Furthermore, if the proof checker is a logic program, the transition from proof checker to proof searcher can be done with minimal changes. For example, let $\mathcal{L}$ be the term denoting a list encoding of the set of literals $\{L_1, \ldots, L_j\}$ and let P and Q be two literals provided by the proof certificate described in Example 4. A logic programming system would then attempt to prove the query

$$\mathit{memb}(P, \mathcal{L}) \wedge \mathit{positive}(P) \wedge \mathit{complement}(P, Q) \wedge \mathit{memb}(Q, \mathcal{L}),$$

where the predicates *positive*, *memb*, and *complement* are all written as, say, first-order Horn clauses in the expected way. If the proof certificate elides the pair $\langle P, Q\rangle$ then it could be asked to prove the query

$$\exists P \exists Q\ [\mathit{memb}(P, \mathcal{L}) \wedge \mathit{positive}(P) \wedge \mathit{complement}(P, Q) \wedge \mathit{memb}(Q, \mathcal{L})].$$

Proving this query is, of course, straightforward and something that a proof checker with unification and backtracking search can easily be expected to perform.

While the proof certificate for all propositional tautologies can be described in terms of LKF in constant size, it does require the proof checker to spend an exponential amount of time to do the checking. Of course, pushing for very small proof certificates can go too far since sometimes communicating "clever" choices can make proofs much easier to check. Consider the formula $(\neg p \vee C) \vee p$

where C is some propositional formula with a large conjunctive normal form. Using the above proof certificate means that there is no way to describe the obvious reason why this formula is tautologous. To allow for more interesting information to be put into proofs, consider the following use of LKF where all propositional connectives are polarized positively.

Example 5. Let B^+ be the result of polarizing positively the connectives of B and let Ξ be an LKF proof for $\vdash \cdot \Uparrow B^+$. It is easy to show that every $\Uparrow$-sequent in Ξ with an empty right-hand-side is of the form $\vdash B^+, \mathcal{L} \Uparrow \cdot$ where $\mathcal{L}$ is a multiset of negative literals. Furthermore, every positive phase starts (reading proofs bottom up) with the *Decide* rule on B and then continues with a series of selections among disjunctions. Proofs using exclusive positive polarizations will, in general, have large decide depths and require larger proofs that those described in Example 4. The additional proof information can, however, make proofs easier to check.

To illustrate the last claim in Example 5, consider a focused proof for the formula $(\neg p \vee^+ C) \vee^+ p$. This formula has a proof of decide depth 2: the first (closest to the root) positive phase is a series of selections in this disjunction that selects $\neg p$ to add to the left of the $\Uparrow$. The second positive phase makes such selections to pick the formula p, and the proof is complete. Notice that the right sequence of choices steers the proof away from considering the subformula C.

Of course, there are many ways to polarize formulas since one can easily mix positive and negative polarizations: these are choices that someone wanting to communicate a proof certificate can make as seems appropriate for the proof objects that they wish to communicate.

To complete this treatment of proof certificates based on consideration of a formula's "matrix," consider the following example of proof certificates deriving their structure from Herbrand's theorem.

Example 6. Herbrand's theorem (see Section 3.1) can be used to validate proof certificates of formulas of the form $\exists \bar{x} B$ (where B is propositional): such certificates can contain a list of substitutions $\theta_1, \ldots, \theta_m$ ($m \geq 1$), all with domain $\bar{x}$, and then a proof certificate for the propositional formula $B\theta_1 \vee \cdots \vee B\theta_m$. Above we discussed various ways to build proof certificates for tautologies. The additional substitution information can be transmitted in the proof certificate as a series of *Decide* rules followed by a series of $\exists$-introduction rules. In general, a proof checker based on logic programming might be expected to recover actual substitution terms so these might be left out of the proof certificate (of course, the number of substitutions m must be supplied).

5 Non-matrix proof systems

A great many proof structures are not based on the "matrix" of formulas. We consider a couple of such proof systems here. Both of these make use of the *cut* inference rules [11]. There are various cut rules for LKF given in [12]: the cut

inference displayed in Figure 1 is an instance of the "key cut" rule while the following inference rule is an instance of the "prime cut" rule:

$$\frac{\vdash \Theta \Uparrow B \qquad \vdash \Theta \Uparrow \neg B}{\vdash \Theta \Uparrow} \; Cut_p$$

Both the "key" and "prime" cut rules can be eliminated in LKF.

Example 7. When a resolution based theorem prover has succeeded in proving a theorem, it has built a resolution dag in which the leaves are clauses, the root is the empty clause (an inconsistency), and the internal nodes are instances of the resolution rule. A *clause* is a closed formula of the form $\forall x_1 \ldots \forall x_n[L_1 \vee \cdots \vee L_m]$ while a *negated clause* is a closed formula of the form $\exists x_1 \ldots \exists x_n[L_1 \wedge \cdots \wedge L_m]$, where $n, m \geq 0$, $\{L_1, \cdots, L_m\}$ a multiset of literals, and $x_1, \ldots, x_n$ is a list of first-order variables. The following predicates are commonly used in building resolution refutations.

1. A clause C is *trivial* if it contains complementary literals.
2. A clause C_1 *subsumes* clause C_2 if there is a substitution instance of the literals in C_1 which is a subset of the literals in C_2.
3. The usual relationship of *resolution* of two clauses C_1 and C_2 to yield C_3 can be characterized by choosing the most general unifier of two complementary literals, one from each of C_1 and C_2. We shall say that C_3 is an *allowed resolvent* if it is constructed by the same rule except that we allow *some* unifier to be used instead of the most general one.

By polarizing clauses using $\vee^-$ (and negated clauses using $\wedge^+$) it is possible to use small LKF proofs to check each of these properties. In particular, it is easy to show that C is trivial if and only if $\vdash \cdot \Uparrow C$ has a proof of decide depth 1. Similarly, by polarizing literals appropriately, C_1 subsumes a non-trivial clause C_2 if and only if $\vdash \neg C_1 \Uparrow C_2$ has a proof of decide depth 1. Finally, C_3 is an allowed resolvent of C_1 and C_2 if and only if $\vdash \neg C_1, \neg C_2 \Uparrow C_3$ has a proof of decide depth 2. It is now a simple matter to take a resolution refutation (also including checks for trivial clauses and subsumption) of the clauses $C_1, \ldots, C_n$ and describe an LKF proof of the sequent $\vdash \neg C_1, \ldots, \neg C_n \Uparrow \cdot$. For example, the following shows how to incorporate into a full proof certificate the fact that C_1 and C_2 yield the resolvent C_{n+1}.

$$\dfrac{\vdash \neg C_1, \neg C_2 \Uparrow C_{n+1} \qquad \dfrac{\vdash \neg C_1, \ldots, \neg C_n, \neg C_{n+1} \Uparrow \cdot}{\vdash \neg C_1, \ldots, \neg C_n \Uparrow \neg C_{n+1}} \; Store}{\vdash \neg C_1, \ldots, \neg C_n \Uparrow \cdot} \; Cut_p$$

By repeating this process, an entire refutation can be converted into an LKF proof with cuts. In all cases, the left premises of all occurrences of the cut rule have small proofs that can be replaced in the final proof certificate with a notation that asks the proof checker to search for proofs up to decide depth 2. In this way, the resulting proof certificate is essentially a direct translation of the refutation and yields a proof certificate that an LKF proof checker can easily

$$\frac{\vdash \Theta\sigma \Uparrow \Gamma\sigma}{\vdash \Theta \Uparrow \Gamma, s \neq t}\ \dagger \qquad \frac{}{\vdash \Theta \Uparrow \Gamma, s \neq t}\ \ddagger \qquad \frac{}{\vdash \Theta \Downarrow t = t}$$

$$\frac{\vdash \Theta \Uparrow \Gamma, B(\nu B)\bar{t}}{\vdash \Theta \Uparrow \Gamma, \nu B\bar{t}} \qquad \frac{\vdash \Theta \Downarrow B(\mu B)\bar{t}}{\vdash \Theta \Downarrow \mu B\bar{t}}$$

Fig. 2. Focused inference rules for $=$ and μ and their duals. The proviso † requires the terms s and t to be unifiable and for σ to be their most general unifier. The proviso ‡ requires that the terms s and t are not unifiable.

check. Of course, the proof checker will have to do some (bounded) proof search to reconstruct the proofs of the left premises of the cut rule.

Example 8. For another example of a proof certificate, we briefly mention the encoding of tabled deduction described in [17]. It is illustrated there that focused proofs (this time using the focused intuitionistic proof system LJF [12]) can be used to capture tabled deduction. There are two important ingredients in the use of tables. First, items that have already been proved need to be available for subsequent items that are still to be proved: that is easily captured using the cut rule in the fashion described above. Second, one must enforce that items in a table must be *reused* and not *reproved*. It was shown in [17] how it is possible to identify negative polarity with atoms that are not in the table and positive polarity with atoms in the table. One also must allow a cut-inference rule that permits the polarity of atoms to switch from negative (on the left premise) to positive (on the right premise). In this way, a table can be translated into an LJF proof with cuts such that every left-premise of a cut has a proof of decide depth 1: one should be able to elide such proofs in a proof certificate.

6 Fixed points and equality

We now extend the first-order logic underlying LKF by adding equality and fixed points and by giving in Figure 2 the introduction rules for these and their duals. Equality $=$ is a positive connective and its De Morgan dual $\neq$ is negative. Similarly, the two fixed points μ and ν are De Morgan duals in which μ is positive and ν is negative. Given that the rules for μ and ν are simply unfoldings of the fixed point, these operators do not yield any particular fixed points. It is possible to have a more expressive proof theory for fixed points that provides also for least and greatest fixed points (see, for example, [4, 5]): in that case, the De Morgan dual of the least fixed point is the greatest fixed point.

Example 9. The following simple logic program defines two predicates on natural numbers, assuming that such numbers are built from zero 0 and successor s.

$$\begin{array}{cc} nat\ 0 \subset true. & leq\ 0\ Y \subset true. \\ nat\ (s\ X) \subset nat\ X. & leq\ (s\ X)\ (s\ Y) \subset leq\ X\ Y. \end{array}$$

The predicate *nat* can be written as the fixed point expression

$$\mu(\lambda p \lambda x.(x = 0) \vee^+ \exists y.(s\ y) = x \wedge^+ p\ y)$$

and binary predicate *leq* (less-than-or-equal) can be written as the expression

$$\mu(\lambda q \lambda x \lambda y.(x = 0) \vee^+ \exists u \exists v.(s\ u) = x \wedge^+ (s\ v) = y \wedge^+ q\ u\ v).$$

In a similar fashion, any Horn clause specification can be made into fixed point specifications (mutual recursions requires standard encoding techniques) that contain *only positive connectives*.

Example 10. Consider proving the sequent

$$\vdash \Theta \Downarrow (leq\ m\ n \wedge^+ N_1) \vee^+ (leq\ n\ m \wedge^+ N_2),$$

where m and n are natural numbers and *leq* is the fixed point expression displayed above. If both N_1 and N_2 are negative formulas, then there are exactly two possible macro rules: one with premise $\vdash \Theta \Uparrow N_1$ when $m \leq n$ and one with premise $\vdash \Theta \Uparrow N_2$ when $n \leq m$ (thus, if $m = n$, both premises are possible). In this way, a macro inference rule can contain an entire Prolog-style computation.

7 Computation and model checking

A traditional approach to leaving out details within a proof is to identify some aspects of the proof as *computation*. An expression to be computed must be communicated in the certificate but the computation trace and the final value do not need to be communicated. When computation is *determinate* (*i.e.*, when expressions have at most one value) this observation has been called the *Poincaré principle* [8, 10]. In rich type systems, such as those found in functional Pure Type System [7], computation is dominated by β-reductions: communicating a proof in that setting does not need to communicate the trace of such β-reductions.

Computation is related to proof certificates in at least three ways. First, the negative phase relates its conclusion to its premises in a determinate fashion: a proof checker simply needs to compute that phase. In Example 4, this phase was determined by the computation of a conjunctive normal form. Second, computation can be inserted *within* an inference rule as is illustrated above in Example 10. In that way, one step in a proof can include arbitrary amounts of computation (all described as a Prolog-like fixed point computation). Third, elided proof details must be reconstructed by a proof-search-style computation and this will also involve computation in the style of logic programming: unification and backtracking can be used to reconstruct elided information.

Besides (determinate) computation, some of the primitives of *model checking* are also naturally captured in this setting of focused proofs. For example, consider the model checking problem of determining if the positive formula $B(x)$ holds for every x that is a member of the set $A = \{a_1, \ldots, a_n\}$. Membership in this set can be encoded as $x = a_1 \vee^+ \cdots \vee^+ x = a_n$ (abbreviated as $A(x)$).

An attempt to prove the sequent $\forall x.A(x) \supset B(x)$ yields the following negative macro rule in LKF. (Here, the implication $C \supset D$ is rendered as $\neg C \vee^- D$.)

$$\frac{\vdash B(a_1) \Uparrow \cdot}{\vdash B(x) \Uparrow x \neq a_1} \quad \cdots \quad \frac{\vdash B(a_n) \Uparrow \cdot}{\vdash B(x) \Uparrow x \neq a_n}$$
$$\vdash \cdot \Uparrow \forall x.[x \neq a_1 \wedge^- \cdots \wedge^- x \neq a_n] \vee^- B(x)$$

In this way, quantification over a finite set is captured precisely as one macro-level inference rules within LKF. The following example illustrates a typical model checking problem.

Example 11. Assume that a label transition system is described by a recursive fixed point expression named $P \xrightarrow{a} P'$ (consider, for example, writing the operational semantics of CCS as a Prolog-like fixed point expression). Simulation in process calculi can be defined as the (greatest) fixed point of the following recursive definition:

$$sim\ P\ Q \equiv \forall P' \forall a[P \xrightarrow{a} P' \supset \exists Q'[Q \xrightarrow{a} Q' \wedge sim\ P'\ Q']].$$

The right-hand side of this definition is composed of *exactly two* macro-rules. The expression $\forall P' \forall a[P \xrightarrow{a} P' \supset \cdot\ ]$ is a negative macro rule since $P \xrightarrow{a} P'$ is positive. The expression $\exists Q'[Q \xrightarrow{a} Q' \wedge^+ \cdot\ ]$ yields a positive macro rule. In this way, the focused proof system is aligned directly with the structure of the actual (model-checking) problem. Notice that if one wishes to communicate a proof of a simulation to a proof checker, no information regarding the use of the negative macro rule needs to be communicated since the proof checker can also perform the computation behind that inference rule (*i.e.*, enumerating all possible transitions of a given process P). Furthermore, eliding proofs of fixed point expressions such as $P \xrightarrow{a} P'$ might also be sensible since the proof checker might well be able to enumerate possible values for some of the values of P, a, and P' when the other values are known [17]. The resulting proof certificate is essentially just the collection of all expressions of the form $sim\ P\ Q$ that are present in the full proof (such a set is also called a *simulation*).

The fact that an entire computation can fit within a macro rule (using purely positive fixed point expressions) provides great flexibility in designing inference rules. Such flexibility allows inference rules to be designed so that they correspond to an "action" within a given computational system. One should note that placing arbitrary computation within an inference rule means that we can have (macro) rules for which their validity is not decidable. Given our interest in proof certificates, however, this does not seem to be a problem in and of itself. A checker may fail to terminate on a given certificate, in which case, we may chose to reject the certificate after waiting some period of time. The engineer of the certificate failed, in this case, to successfully communicate a proof. The certificate structure might then need to be redesigned.

Example 12. The Lucas-Lehmer theorem states that n is prime if and only if there exists an integer a such that $1 < a < n$ and $a^{(n-1)} \equiv 1 \pmod{n}$ and

for all prime factors q of $n-1$, it is the case that $a^{(n-1)/q} \not\equiv 1 \pmod{n}$. This theorem can be used to build certificates of primality [20]: proof certificates of the same claim are also easy to develop. Assume that the Lucas-Lehmer theorem has already been proved and placed into a (trusted) library. In principle, the certificate proving primality of n involves producing the witness a and the prime factors of $n-1$. The rest of this certificate requires various straightforward computations (via fixed points) as well as proof certificates that show that the factors q of $n-1$ are, indeed, primes.

8 Related Work

Proof carrying code [19] was an earlier attempt at producing, communicating, and checking proof objects. Much of that effort was focused on theorems involving assertions about mobile and imperative programs in a setting where proof objects were often highly optimized in order to be used in resource-limited systems. Shankar uses proof certificates as a part of a prover architecture in which the claims of untrusted inference procedures are validated by communicating certificates that are then checked by checkers that have been verified relative to a small kernel checker [21]. We are concerned here, however, with all manner of formal proof objects from a much wider range of applications with no a priori restriction on resources.

The Dedukti [9] proof checker shares some characteristics with our proposal for proof checking. Instead of using proof theory and focusing, Dedukti employs *deduction modulo* [10] as a framework for building large scale inference rules from theories and smaller inferences. This system also separates computation steps from deduction steps: in the case of Dedukti, computation is, however, determinant and is based on functional programming-style computation. Proofs are not permitted to contain holes and proof search for proof reconstruction is not available.

9 Future work

This proposal can be developed along a number of directions.

Proof reconstruction when equality is a logical connective Proof reconstruction in first-order logics requires unification. When one introduces equality as a logical connective, as we did in Figure 2, proof reconstruction must deal with unification for elided terms ("logic variables") as well as eigenvariables. Standard procedures for (higher-order) unification work well when only one such class of variables is present: unification with these two classes of variables is still to be developed.

Induction and co-induction Baelde has investigated focused proof system that incorporate both induction and co-induction [4, 5]. An experimental prover [6] has demonstrated that small focused proofs involving induction can be proved completely automatically, leaving open the possibility of proof reconstruction even for proof certificates that leave out subproofs of simple inductive lemmas.

Combining intuitionistic and classical logics One does not want to have to deal with two sets of different proof theories: one each for classical and intuitionistic logics. Ideally, these should be combined into one logic and (focused) proof system: see [13] for an initial proposal for such a logic.

Counterexamples and partial proofs Structural proof theory deals with complete proofs. In a setting where proofs are developed in a distributed setting and employs an array of theorem proving technologies, partial proofs (proofs with unproved premises) become important objects that should be studied properly. Similarly, counterexamples are extremely valuable documents that should be formally included in a comprehensive approach to proof certificates. These two concepts should also be tied together with techniques similar to those used to eliminate cuts. For example, when someone finds a counterexample to an open premise of a partial proof, one would like to systematically explore how much of the partial proof needs to be rewound in order to avoid that counterexample.

Building and trusting proof checkers Significant computational resources and flexibility are need for proof checking and proof reconstruction. The Dedukti proof checker [9] places Haskell into its trusted code base. Our framework here is better served by logic programming. Of course, such logic programming systems must be logically sound in the strongest senses. Since there are bindings within formulas (quantifiers) and bindings within proofs (eigenvariables), the λProlog programming language, which treats bindings entirely declaratively, might make a good implementation language for proof checkers. Of course, this means that a λProlog implementation, such as Teyjus [18], must enter the trusted core. Accepting higher-order logic programming languages into the "trusted base of code" is, in fact, a familiar theme: both λProlog and Twelf have been proposed as part of the trusted code base for supporting proof-carrying code [3].

10 Conclusion

We have overviewed a *foundational* approach to designing proof certificates that satisfies the following four desiderata (described in more depth in [15]): they should be (*i*) checkable by simple proof checkers, (*ii*) flexible enough that existing provers can conveniently produce such certificates from their internal evidence of proof, (*iii*) directly related to proof formalisms used within the structural proof theory literature, and (*iv*) permit certificates to elide some proof information with the expectation that a proof checker can reconstruct the missing information using bounded and structured proof search. Central to our design of such proof certificates is the proof-theoretic notion of focused proof system.

Acknowledgments. I wish to thank Alberto Momigliano, Gopalan Nadathur, Alwen Tiu, and the referees for their comments on an earlier draft of this paper.

References

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

2. Peter B. Andrews. Theorem-proving via general matings. *J. ACM*, 28:193–214, 1981.
3. Andrew W. Appel and Amy P. Felty. Polymorphic lemmas and definitions in λProlog and Twelf. *Theory and Practice of Logic Programming*, 4(1-2):1–39, 2004.
4. David Baelde. *A linear approach to the proof-theory of least and greatest fixed points*. PhD thesis, Ecole Polytechnique, December 2008.
5. David Baelde. Least and greatest fixed points in linear logic. Accepted to the *ACM Transactions on Computational Logic*, September 2010.
6. David Baelde, Dale Miller, and Zachary Snow. Focused inductive theorem proving. In J. Giesl and R. Hähnle, editors, *Fifth International Joint Conference on Automated Reasoning*, number 6173 in LNCS, pages 278–292, 2010.
7. Henk Barendregt. Lambda calculus with types. In S. Abramsky, Dov M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 117–309. Oxford University Press, 1992.
8. Henk Barendregt and Erik Barendsen. Autarkic computations in formal proofs. *J. of Automated Reasoning*, 28(3):321–336, 2002.
9. Mathieu Boespflug. *Conception d'un noyau de vérification de preuves pour le λΠ-calcul modulo*. PhD thesis, Ecole Polytechnique, 2011.
10. Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Theorem proving modulo. *J. of Automated Reasoning*, 31(1):31–72, 2003.
11. Gerhard Gentzen. Investigations into logical deductions. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131. North-Holland, Amsterdam, 1969. Translation of articles that appeared in 1934-35.
12. Chuck Liang and Dale Miller. Focusing and polarization in linear, intuitionistic, and classical logics. *Theoretical Computer Science*, 410(46):4747–4768, 2009.
13. Chuck Liang and Dale Miller. Kripke semantics and proof systems for combining intuitionistic logic and classical logic. Submitted, September 2011.
14. Per Martin-Löf. Constructive mathematics and computer programming. In *Sixth International Congress for Logic, Methodology, and Philosophy of Science*, pages 153–175, Amsterdam, 1982. North-Holland.
15. Dale Miller. Communicating and trusting proofs: The case for broad spectrum proof certificates. Available from author's website, June 2011.
16. Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.
17. Dale Miller and Vivek Nigam. Incorporating tables into proofs. In J. Duparc and T. A. Henzinger, editors, *CSL 2007: Computer Science Logic*, volume 4646 of *LNCS*, pages 466–480. Springer, 2007.
18. 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.
19. George C. Necula. Proof-carrying code. In *Conference Record of the 24th Symposium on Principles of Programming Languages 97*, pages 106–119, Paris, France, 1997. ACM Press.
20. Vaughan R. Pratt. Every prime has a succinct certificate. *SIAM Journal on Computing*, 4(3):214–220, September 1975.
21. Natarajan Shankar. Trust and automation in verification tools. In S. D. Cha, J-Y. Choi, M. Kim, I. Lee, and M. Viswanathan, editors, *ATVA: Automated Technology for Verification and Analysis*, volume 5311 of *LNCS*, pages 4–17. Springer, 2008.