Foundational Proof Certificates

Dale Miller

INRIA-Saclay and LIX/École Polytechnique
dale.miller@inria.fr

1 Introduction

Consider a world where exporting proof evidence into a well defined, universal, and permanent format is taken as "feature zero" for computational logic systems. In such a world, provers will communicate and share theorems and proofs; libraries will archive and organize proofs; and marketplaces of proofs would be open to any prover that admits checkable proof objects. In that world, proof checkers will be the new gatekeepers: they will be entrusted with the task of checking that claimed proof evidence elaborates into a formal proof.

Logicians and proof theorists have worked on defining notions of proof that are not based on technology and do not have version numbers attached to them. There are many such proof systems in the literature: Hilbert-Frege proofs, Gentzen's sequent calculus proofs, Prawitz's natural deduction proofs, etc. Each of these proof systems have been given precise syntax and meaning. While such well studied proof descriptions exist, a quick review of the current state of automated and interactive theorem provers reveals that provers seldom output their "proof evidence" using such proof systems. While there is a lot of interest in having provers share and trust each other's proofs (see, for example, [3, 10, 28]) most of that work has been based on building bridges between two specific provers: a change in the version number of one prover can cause that bridge to collapse.

The ProofCert project [22] has as one of its goals the development of a flexible framework for defining the semantics of a wide range of proof evidence in such a way that provers would define the meaning of their own proof evidence and trusted proof checkers would be able to interpret that meaning and check its formal correctness. To achieve this goal, we must first be able to separate proof evidence from its provenance and then provide a formal and clear framework for defining the semantics of proof evidence. The ProofCert project is focused on the problem of checking *formal* proof: there is no assumption made that such formal proofs are actually readable by humans.

2 Defining the semantics of proof evidence

The wide range of provers in use today represent their "proof evidence" in many different ways. Such evidence might be resolution refutations, sets of links between atoms in a formula, natural deduction proofs, typed λ-terms, or proof scripts. If we insist that provers output their proof evidence as a document that

could be transmitted and formally checked, we immediately face the problem that there will necessarily be many different languages and structures describing proof evidence. How can we deal with such a plurality of proof languages?

Similar situations have, of course, appeared and been addressed within computer science. For example, consider the problem of defining the static and dynamic semantics of programming languages. Sometimes, a particular implementation of a given programming language actually defines that language's syntax and semantics. Such an ad hoc notion of language definition has largely been replaced by the use of formal frameworks where syntax and semantics are defined in a universal and permanent (*i.e.*,, technology independent) fashion. For example, today, it is common that a programming language's syntax is given using the formal framework of grammars. Using grammars, one describes "declaratively" how to generate the set of legal programming language expressions. Advantages of such a system are numerous: anyone can now implement their own parser while knowing the specification of what they need to implement. Furthermore, people can now attempt to automate the entire process of producing parsers from grammars: in this way, the correctness of many parsers can be reduced to the problem of establishing the correctness of the parser generator. The use of such a framework does come with some costs. For example, context free grammars can be ambiguous and it is undecidable in general to tell if a given such grammar is ambiguous. Also, parser generators usually support restricted sets of grammars (*e.g.*, LALR(1), LR(k)) and the syntax of a given programming language might have to be simplified to conform to various restrictions imposed by generators.

Similar observations also hold for the problem of defining the dynamic semantics of a programming language. It is common place (at least in the research community) to formally define the semantics of a programming language using the "declarative" techniques found in operational semantics [16, 31]. Using such semantic specifications it is possible to define a programming language precisely enough that various compilers and interpreters can be build for the same language [27] and for formal theorems to be proved about them.

In the rest of this paper, we shall describe the *foundational proof certificate* framework that can be used to define the semantics of a wide range of proof evidence.

3 The chemistry of inference

The foundational approach to proof certificates is a framework where large inference rules are built from small inference rules. In particular, we first identify the "atoms" of inference and the "rules of chemistry" that then allow us to build the "molecules" of inference.

3.1 The atoms: sequent calculus inference rules

The smallest elements of inference that we consider come from Gentzen's sequent calculus [11]: we shall assume that the reader is familiar with the basics of sequent calculus.

While the foundational proof certificate framework can be described for both classical and intuitionistic first order logics, we restrict our attention here to just classical logic in order to simplify our presentation. Given this simplification, we can also assume that formulas are placed into negation normal form (*i.e.*, negations have only atomic scope) and if B is not atomic, the expression $\neg B$ is meant to denote the formula that results from using de Morgan dualities to push the outermost negation in over all connectives. We can also limit ourselves to using only one-sided sequents, written $\vdash \Delta$. Here, Δ is a collection of formulas: Gentzen used lists of formulas exclusively but we shall use multisets instead.

The inference rules of Gentzen's sequent calculus can be divided into three groups: *structural rules*, *identity rules*, and *introduction rules*. We shall consider these rules as they are applied to one-sided sequents.

The structural rules are the familiar rules of *exchange*, *weakening*, and *contractions*. We shall not use the exchange rule here since it is meaningless when used with multisets of formulas. Ultimately, weakening and contraction will not be separate rules but will be built into other rules.

There are two identity rules, namely, the *initial rule* and the *cut rule*:

$$\frac{\vdash \Delta_1, B \quad \vdash \Delta_2, \neg B}{\vdash \Delta_1, \Delta_2}\ cut \qquad\qquad \frac{}{\vdash B, \neg B, \Delta}\ init$$

These rules are collectively called "identity" rules since they both involve checking that a formula is identical to the negation normal form of another formula. Note that the weakening rule is built into the *init* rule. Gentzen proved that all instances of these identity rules can be eliminated except for instances of the *init* rule where B is an atomic formula.

The introduction rules give meaning to the logical connectives. In the version of classical logic we are considering, the only logical connectives are $\wedge$, $\vee$, $\forall$, and $\exists$. We shall take the following familiar rules for the quantifiers:

$$\frac{\vdash \Delta, [y/x]B}{\vdash \Delta, \forall x.B} \quad \text{and} \quad \frac{\vdash \Delta, [t/x]B}{\vdash \Delta, \exists x.B}\ .$$

For the propositional connectives, we find different possibilities. For example, the one-sided sequent calculus rules most closely related to Gentzen's two-sided sequent calculus rules would be

$$\frac{\vdash \Delta, B \qquad \vdash \Delta, C}{\vdash \Delta, B \wedge C}\ \wedge\text{-}I_a \qquad \frac{\vdash \Delta, B}{\vdash \Delta, B \vee C}\ \vee\text{-}I_a \qquad \frac{\vdash \Delta, C}{\vdash \Delta, B \vee C}\ \vee\text{-}I_a.$$

Other natural possibilities exist:

$$\frac{\vdash \Delta_1, B \qquad \vdash \Delta_2, C}{\vdash \Delta_1, \Delta_2, B \wedge C}\ \wedge\text{-}I_m \qquad \frac{\vdash \Delta, B, C}{\vdash \Delta, B \vee C}\ \vee\text{-}I_m.$$

As we have learned from linear logic [12], the inference rules for conjunction and disjunction (and their units) can come in two forms: the additive rules (displayed above with the subscript a) and the multiplicative rules (displayed above with the subscript m).

It is the case, of course, that when both contraction and weakening are available, the additive and the multiplicative versions of these rules are interchangeable: as a result, most sequent calculus systems select one version of these rules only. For example, many papers dealing with theorem proving in classical logic commonly use the $\wedge$-I_a and $\vee$-I_m rules since they are invertible.

In our case, we are striving to collect a good set of atomic inferences and we are helped if we can allow for having both additive and multiplicative versions of these inference rules. We shall use the following technique to disambiguate when these rules are applied. First, we introduce two versions of the conjunction $\wedge^+$, $\wedge^-$ and two versions of the disjunction $\vee^+$, $\vee^-$. Second, we write the additive and multiplicative inference rules as

$$\frac{\vdash \Delta, B \qquad \vdash \Delta, C}{\vdash \Delta, B \wedge^- C} \qquad \frac{\vdash \Delta, B}{\vdash \Delta, B \vee^+ C} \qquad \frac{\vdash \Delta, C}{\vdash \Delta, B \vee^+ C}$$

$$\frac{\vdash \Delta_1, B \qquad \vdash \Delta_2, C}{\vdash \Delta_1, \Delta_2, B \wedge^+ C} \qquad \frac{\vdash \Delta, B, C}{\vdash \Delta, B \vee^- C}$$

Note that the rules for the negative connectives $\wedge^-$, $\vee^-$ are invertible. (For the sake of completeness, we introduce the polarized forms for the true and false constants t^-, t^+, f^-, f^+ in the next section.) Third, if B is a first-order formula (an "unpolarized" formula), we shall write $\hat{B}$ to denote any ("polarized") formula that results from replacing all occurrences of $\wedge$ and $\vee$ in B with one of the signed versions of the corresponding connective. If B contains n occurrences of either conjunction and disjunction, then $\hat{B}$ ranges overs the 2^n polarized forms of B.

3.2 The chemistry of inference: focused proof systems

While the sequent calculus rules capture tiny steps in deduction, they are poorly equipped to capture larger scale notions of inference. What is needed is a means of organizing these small rules into larger and more familiar rules. For example, the work on *uniform proofs* in the late 1980's [25, 26] provided a restriction on sequent calculus proofs that allowed such proofs to capture the alternating phases of goal-reduction and backchaining that take place within logic programming proof search. While the technique of uniform proofs provided a description of the simple structure of inference in logic programming, it was not flexible enough to capture many other forms of inference in other computational logic systems. As Andreoli [1] showed, when one moves the alternating phase structure of uniform proofs to linear logic, a much more flexible means for structuring proofs arises. Andreoli called his new proof system a *focused* proof system. Comprehensive focused proof systems for classical and intuitionistic logic were later introduced by Liang and Miller [17, 18]. We shall now present more details of LKF [18], a focused proof system for classical logic.

$$\frac{}{\vdash \Theta \Downarrow t^+} \qquad \frac{\vdash \Theta \Downarrow B_1 \quad \vdash \Theta \Downarrow B_2}{\vdash \Theta \Downarrow B_1 \wedge^+ B_2} \qquad \frac{\vdash \Theta \Downarrow B_i \quad i \in \{1,2\}}{\vdash \Theta \Downarrow B_1 \vee^+ B_2} \qquad \frac{\vdash \Theta \Downarrow [t/x]B}{\vdash \Theta \Downarrow \exists x.B}$$

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

$$\frac{\vdash \Theta \Uparrow [y/x]B, \Gamma}{\vdash \Theta \Uparrow \forall x.B, \Gamma} \qquad \frac{\vdash \Theta \Uparrow B \quad \vdash \Theta \Uparrow \neg B}{\vdash \Theta \Uparrow \cdot}\ cut \qquad \frac{}{\vdash \neg P_a, \Theta \Downarrow P_a}\ init$$

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

Here, P is a positive formula; N is a negative formula; P_a is a (positive) atom; and C is a positive formula or negated atom. Also, y is a variable that is not free in any formula in the conclusion sequent of the $\forall$ introduction rule.

Fig. 1. LKF rules

We shall call a polarized formula *positive* if it is either atomic, or its top-level connective is $\wedge^+$, $\vee^+$, t^+, f^+, or $\exists$. Similarly, a polarized formula is *negative* if it is either a negated atom, or its top-level connectives is $\wedge^-$, $\vee^-$, t^-, f^-, or $\forall$.

The inference rules for *LKF* are given in Figure 1. Note that these inference rules involve sequents of the form $\vdash \Theta \Uparrow \Gamma$ and $\vdash \Theta \Downarrow B$ where Θ is a multiset of formulas, Γ is a list of formulas, and B is a formula (all formulas are polarized formulas). Such sequents can be approximated as the one-sided sequents $\vdash \Theta, \Gamma$ and $\vdash \Theta, B$, respectively. Furthermore, introduction rules are applied to either the first element of the list Γ in the $\Uparrow$ sequent or the formula B in the $\Downarrow$ sequent. This occurrence of the formula B is called the *focus* of that sequent. Proofs in *LKF* are built using two kinds of alternating *phases*. The *negative* phase is composed of invertible inference rules and only involves $\Uparrow$-sequents in the conclusion and premise. The other phase is the *positive* phase: here, applications of such inference rules often require choices. In particular, the introduction rule for the disjunction requires selecting either the left or right disjunct and the introduction rule for the existential quantifier requires selecting a term for instantiating the quantifier. The initial rule can terminate a positive phase and the cut rule can restart a negative phase. Finally, there are three structural rules in *LKF*. The *store* rule recognizes that the first formula to the right of the $\Uparrow$ is either a negative atom or a positive formula: such a formula does not have an invertible inference rule and, hence, its treatment is delayed by storing it on the left. The *release* rule is used when the formula under focus (*i.e.*, the formula to the right of the $\Downarrow$) is no longer positive: at such a moment, the phase changes to the negative phase. Finally, the *decide* rule is used at the end of the negative phase to start a positive phase by selecting a previously stored positive formula as the new focus. Note that the contraction rule is built into the decide rule and that the context Θ is treated additively even by the multiplicative connective $\wedge^+$.

It is proved in [18] that if B is a classical theorem and $\hat{B}$ is any polarization of B, then $\vdash \cdot \Uparrow \hat{B}$ has an *LKF* proof. Conversely, if $\vdash \cdot \Uparrow \hat{B}$ is provable in *LKF* then

B is a theorem. Thus the different polarizations do not change *provability* but can radically change the structure of proofs. A simple induction on the structure of an *LKF* proof of $\vdash \cdot \Uparrow B$ (for some polarized formula B) reveals that every formula that occurs to the left of $\Uparrow$ or $\Downarrow$ in one of its sequents is either a negated atom or a positive formula. Also, it is immediate that the only occurrence of a contraction rule is within the decide rule: thus, only the positive formulas are contracted. Since there is flexibility in how formulas are polarized, the choice of polarization can, at times, lead to greatly reduced opportunities for contraction. When one is able to eliminate or constrain contractions, naive proof search can sometimes become a decision procedure.

It is the negative and positive phases that are the *macro* or *synthetic* inference rules: these are the molecules of inference and are built from the atoms of inference. Note that phases are organized as follows:

$$\dfrac{\dfrac{\vdash \Theta_{i1} \Uparrow \cdot \quad \cdots \quad \vdash \Theta_{ij} \Uparrow \cdot}{\dfrac{\vdash \Theta \Uparrow N_i}{\vdash \Theta \Downarrow N_i}\ \mathit{release}} \quad \cdots}{\dfrac{\vdash \Theta \Downarrow P}{\vdash \Theta \Uparrow \cdot}\ \mathit{decide}} \quad \cdots$$

Specifically, a positive phase has as its root a decide rule and as leaves the conclusions of release rules, while the negative phase ends with a sequent that is the conclusion of a decide rule. Together, a negative phase above a positive phase (a pair sometimes called a *bi-pole*) has a conclusion of the form $\vdash \Theta \Uparrow \cdot$ and premises of the form $\vdash \Theta' \Uparrow \cdot$ where Θ is a sub-multiset of Θ'. Thus, the sequence of decide and release rules determine boundaries between phases. A phase may also end, of course, with the introduction rules for t^- and t^+ and the initial rule.

3.3 The engineering of inference rules

To illustrate the possibilities allowed by *LKF*, we now consider a couple of different approaches to building *LKF* proofs for a propositional tautology B.

One approach involves picking $\hat{B}$ to be the polarized version of this formula that contains only the negative connectives t^-, f^-, $\wedge^-$, $\vee^-$. In this case, the only *LKF* proofs of $\vdash \cdot \Uparrow \hat{B}$ have exactly one negative phase: it has as a conclusion $\vdash \cdot \Uparrow \hat{B}$ and has a possibly exponential number (in the size of B) of premises of the form $\vdash L_1, \ldots, L_n \Uparrow \cdot$, where $L_1, \ldots, L_n$ are literals (atoms or negated atoms). Note that the negative phase is completely determinate (computed functionally from $\hat{B}$). The premises of this negative phase are of the form $\vdash L_1, \ldots, L_n \Uparrow \cdot$. The only way to prove such a sequent in *LKF* (without cut) is to use one occurrence each of the decide and init rules.

Note that this use of *LKF* and negative polarities leads to the following *protocol* for proving a tautology: first use negative rules to compute essentially the conjunctive normal form of B and then show that there are complementary pairs in each of the remaining premises. Determining these complementary pairs could

be done by having some external source of information (like an oracle) provide the right answer or by a search. Such a search would be rather shallow and easily performed in a complete fashion. Thus, such a protocol could be used to define a simple kind of *proof certificate*: the certificate would announce that the negative connectives are to be used and that the complementary pair of literals in each premise is either explicitly listed in that certificate or that the certificate checker should do the search for complementary literals. If the certificate provides the complementary pairs, then the certificate would be exponentially large (based on the size of B) or it would be constant sized. In either case, the checking time for this certificate would be exponential since checking involves computing the conjunctive normal form of B.

Consider the appropriateness of such an approach for showing that the formula $B = (p \vee (C \vee \neg p))$ is a tautology: here p is a propositional constant and C is a possibly large propositional formula. Clearly, this formula is tautologous. While using the protocol above to prove this formula would work, it is easy to describe a more direct proof, one where we would like to insert some "clever" information into the proof building stage. To do this, we use the positive connectives t^+, f^+, $\wedge^+$, $\vee^+$. The "clever" choices are injected twice into the proof with the mark †. The subformula C is avoided in this proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash B, \neg p \Downarrow p}\, init}{\vdash \hat{B}, \neg p \Downarrow (p \vee^+ \hat{C}) \vee^+ \neg p}\, \dagger}{\vdash \hat{B}, \neg p \Uparrow \cdot}\, decide}{\vdash \hat{B} \Uparrow \neg p}\, store}{\vdash \hat{B} \Downarrow \neg p}\, release}{\vdash \hat{B} \Downarrow (p \vee^+ \hat{C}) \vee^+ \neg p}\, \dagger}{\vdash \hat{B} \Downarrow \cdot}\, decide$$

Clearly, different polarities can lead to rather different disciplines for organizing proofs. The negative phase does not listen to any outside oracles: instead, it simply performs a (determinate) computation that carries a concluding sequent to a list of premises. On the other hand, the positive phase consumes information—such as which branch of a disjunction or which instance of an existential quantifier to consider. That information can be supplied, in principle, from either an oracle (by reading a proof certificate) or a non-deterministic search. We now formalize more carefully how to integrate a proof certificate with focused proof construction.

3.4 Clerks, experts, certificates, and indexes

In order to translate the information in a given proof certificate into instructions to drive the kernel's (that is, *LKF*'s) inference rules, we use the notion of *clerks and experts* [7, 8]. An analogy can help motivate our proposed design. Imagine an accounting office that needs to check that a certain collection of financial

$$\frac{true_e(\Xi)}{\Xi \vdash \Theta \Downarrow t^+} \qquad \frac{\Xi_1 \vdash \Theta \Downarrow B_1 \quad \Xi_2 \vdash \Theta \Downarrow B_2 \quad \wedge_e(\Xi, \Xi_1, \Xi_2)}{\Xi \vdash \Theta \Downarrow B_1 \wedge^+ B_2}$$

$$\frac{\Xi' \vdash \Theta \Downarrow B_i \qquad i \in \{1,2\} \qquad \vee_e(\Xi, \Xi', i)}{\Xi \vdash \Theta \Downarrow B_1 \vee^+ B_2} \qquad \frac{\Xi' \vdash \Theta \Downarrow [t/x]B \qquad \exists_e(\Xi, \Xi', t)}{\Xi \vdash \Theta \Downarrow \exists x.B}$$

$$\frac{\Xi' \vdash \Theta \Uparrow \Gamma \quad f_c(\Xi, \Xi')}{\Xi \vdash \Theta \Uparrow f^-, \Gamma} \qquad \frac{\Xi' \vdash \Theta \Uparrow A, B, \Gamma \quad \vee_c(\Xi, \Xi')}{\Xi \vdash \Theta \Uparrow A \vee^- B, \Gamma}$$

$$\frac{}{\Xi \vdash \Theta \Uparrow t^-, \Gamma} \qquad \frac{\Xi_1 \vdash \Theta \Uparrow A, \Gamma \quad \Xi_2 \vdash \Theta \Uparrow B, \Gamma \quad \wedge_c(\Xi, \Xi_1, \Xi_2)}{\Xi \vdash \Theta \Uparrow A \wedge^- B, \Gamma}$$

$$\frac{\Xi' \vdash \Theta \Uparrow [y/x]B, \Gamma \qquad \forall_c(\Xi, \Xi') \qquad y \text{ not free in } \Xi, \Theta, \Gamma, B}{\Xi \vdash \Theta \Uparrow \forall x.B, \Gamma}$$

$$\frac{\Xi' \vdash \Theta, \langle l, C\rangle \Uparrow \Gamma \quad store_e(\Xi, C, \Xi', l)}{\Xi \vdash \Theta \Uparrow C, \Gamma}\ store$$

$$\frac{\Xi_1 \vdash \Theta \Uparrow B \qquad \Xi_2 \vdash \Theta \Uparrow \neg B \qquad cut_e(\Xi, \Theta, \Xi_1, \Xi_2, B)}{\Xi \vdash \Theta \Uparrow \cdot}\ cut$$

$$\frac{\Xi' \vdash \Theta \Uparrow N \quad release_e(\Xi, \Xi')}{\Xi \vdash \Theta \Downarrow N}\ release \qquad \frac{init_e(\Xi, \Theta, l) \quad \langle l, \neg P_a\rangle \in \Theta}{\Xi \vdash \Theta \Downarrow P_a}\ init$$

$$\frac{\Xi' \vdash \Theta \Downarrow P \quad decide_e(\Xi, \Theta, \Xi', l) \quad \langle l, P\rangle \in \Theta \quad \text{positive}(P)}{\Xi \vdash \Theta \Uparrow \cdot}\ decide$$

Fig. 2. LKF^a: *LKF* augmented with premises that invoke clerks and experts.

documents represents a legal transaction. The office workers called experts are given the responsibility of looking into the collection and extracting information: they must *decide* into which series of transactions to dig and they need to know when to *release* their findings for storage and later reconsideration. On the other hand, the clerks are responsible for taking information released by the experts and performing some computations on them, including their *indexing* and *storing*. Of course, the division of labor between experts and clerks arises from the different characteristics of the positive and negative phases of proof structures in *LKF*.

The inference rules in Figure 2 define LKF^a, the *augmented LKF* proof system. The augmentation adds three kinds of objects to *LKF*. The first is the actual proof certificate as a term: the syntactic variable Ξ ranges over such certificates. The second addition is the extra premises to all inference rules. The positive inference rules have calls to *experts*: these are predicates that know how to extract information from proof certificates in order to supply information required by a positive inference rule. Thus, the expert for the existential induction rule, $\exists_e(\Xi, \Xi', t)$ is supposed to hold when the certificate Ξ indicates that the term t

can be used to instantiate the corresponding $\exists$ quantifier: once that information has been extracted, the remaining certificate is Ξ'.

The third item that we need to add to the LKF^a proof system was hinted at with the office analogy above: when a clerk releases some information to be considered later, that item must be stored. Storing must of course support "recall" (embodied by the decide inference rule). To do such store and recall flexibly, we shall allow the office workers to agree on an actual indexing scheme for stored formulas. Such index schemes can be various. For example, we have found all the following indexes useful in different styles of proof evidence: the formula itself, a de Bruijn number, a formula occurrence, a link name in a proof net, a line number in a Frege proof, and a clause number in a resolution refutation. Note that in LKF^a, the context Θ does not denote a multiset of formulas but rather a multiset of pairs $\langle l,C\rangle$ where l is an index and C is a formula. It is the responsibility of the store clerk to compute an index for the formula that is moved from the right to the left of the $\Uparrow$. Similarly, the decide rule selects a formula from the Θ context by providing an index.

It now is clear what a proof certificate must contain. First, it must describe both the datatypes used to build certificates and indexes. Second, it must provide a method of polarizing a formula B into $\hat{B}$. Third, it must provide the specification of the various clerks and experts. These can be described either as inference rules themselves or, equivalently, as simple logic programs. Note that the rules in LKF^a are always sound, no matter how one specifies the clerks and experts. This soundness is an important feature for a checker that must be trusted even though significant specifications (and code) are supplied from outside the kernel.

3.5 Examples of clerks and experts

We present here two examples of how one can specify clerks and experts. Given our presentation to this point, it will probably be difficult to understand the details of how these specifications work. More details can be found in [8] from where these examples are taken. Here, we will limit ourselves to some observations. We shall also use λProlog syntax to provide the specifications of clerks and experts: inference rules could have also been used but the λProlog syntax is more compact (see also the discussion in Section 4).

`cnf : cert`	`idx : form -> index`
$\forall C.\ \mathit{store}_c(\mathtt{cnf}, C, \mathtt{cnf}, \mathtt{idx}(C))$.	$\wedge_c(\mathtt{cnf}, \mathtt{cnf}, \mathtt{cnf})$.
$\forall \Theta \forall l.\ \mathit{init}_e(\mathtt{cnf}, \Theta, l)$.	$\vee_c(\mathtt{cnf}, \mathtt{cnf})$.
$\forall \Theta \forall l.\ \mathit{decide}_e(\mathtt{cnf}, \Theta, \mathtt{cnf}, l)$.	$f_c(\mathtt{cnf}, \mathtt{cnf})$.
$\mathit{release}_e(\mathtt{cnf}, \mathtt{cnf})$.	

Fig. 3. A checker based on a simple decision procedure

A decision procedure as a proof certificate. Figure 3 presents a concrete specification of the decision procedure for propositional formulas described in Section 3.3. Figure 3 first lists the constructors for certificates (type `cert`) and indexes (type `idx`). In this case, there is a unique inhabitant of type `cert` and that is just the name of this decision procedure (that is, this part of the certificate contains no information). Similarly, the only way to build an index is to use the formula itself (thereby trivializing the indexing mechanism here). Since we are assuming that conjunction and disjunction are polarized negatively, only clerks are associated to introduction rules. Finally, the expert predicates for the initial rule and the decide rule are not acting like experts at all: they formally allow any context Θ and any index l to be related to the `cnf` certificate. Such behavior is fine, however, since the rules in Figure 2 make additional checks on Θ and l and these checks actually discover complementary pairs of literals.

This certificate illustrates an important aspect of our proposal for FPC: some detail from a proof can, in principle, be elided and this may not cause a problem for proof checking. In the case of this certificate format, there might be several proofs of a sequent containing just literals, since it might contain many different complementary pairs. One could rewrite this certificate format to explicitly contain a *mating*, that is a set of pairs of complementary literals that spans all such clauses [2]. Such a mating is, of course, possibly exponentially large with respect to the tautology being checked. But if we allow for some search, we can do some "proof reconstruction" that involves searching for complementary pairs. Allowing such reconstruction makes it possible for this FPC to have a constant size instead of the possibly exponential size for recording an explicit mating.

Resolution refutations as proof certificates. Figure 4 lists the constructors and the clerks and experts that can be used to specify the semantics of a simple form of resolution refutation. There are two key parts of this checker. First, if two clauses C_1 and C_2 resolve to yield clause C_0, then there is an LKF proof of $\vdash \neg C_1, \neg C_2 \Uparrow C_0$ that has decide depth 3 or less (the decide depth of a proof is the maximum number of decide rules along a path in that proof). The first set of specifications in Figure 4 describe how the clerks and experts can be specified to check for the existence of such small proofs. Second, clauses are indexed by natural numbers and the resolution refutation is a list of triples. Each of these triples are checked using the specification above to confirm that it is a valid binary resolution and then the cut-rule is used to integrate the resolvent into the other clauses. The second set of clauses specify clerks and experts that direct the LKF^a kernel to trace out a proof whose backbone is a series of cuts all of whose left premises are the small proofs that are responsible for checking claimed binary resolutions. For more explanation about this certificate, see [8, 23].

4 A reference checker

Logic programming can sometimes be used to convert a declarative specification into a prototype implementation. For example, versions of Prolog can be used to

```
idx : int -> index                lit : form -> index
dl  : list int -> cert          ddone : cert
```

$$\forall L.\ \vee_c(\texttt{dl}(L), \texttt{dl}(L)). \qquad \forall L.\ true_e(\texttt{dl}(L)).$$
$$\forall L.\ f_c(\texttt{dl}(L), \texttt{dl}(L)). \qquad \forall L.\ \forall_c(\texttt{dl}(L), \texttt{dl}(L)).$$
$$\forall C \forall L.\ store_c(\texttt{dl}(L), C, \texttt{dl}(L), \texttt{lit}(C)). \qquad \forall L.\ \exists_e(\texttt{dl}(L), \texttt{dl}(L), T).$$
$$\forall L \forall P \forall \Theta.\ decide_e(\texttt{dl}(L), \Theta, \texttt{ddone}, \texttt{lit}(P)). \qquad \forall L.\ \wedge_e(\texttt{dl}(L), \texttt{dl}(L), \texttt{dl}(L)).$$
$$\forall I \forall \Theta.\ decide_e(\texttt{dl}([I]), \Theta, \texttt{dl}([]), \texttt{idx}(I)). \qquad \forall l \forall \Theta.\ init_e(\texttt{ddone}, \Theta, l).$$
$$\forall I \forall J \forall \Theta.\ decide_e(\texttt{dl}([I, J]), \Theta, \texttt{dl}([J]), \texttt{idx}(I)). \qquad \forall l \forall L \forall \Theta.\ init_e(\texttt{dl}(L), \Theta, l).$$
$$\forall I \forall J \forall \Theta.\ decide_e(\texttt{dl}([J, I]), \Theta, \texttt{dl}([J]), \texttt{idx}(I)). \qquad \forall L.\ release_e(\texttt{dl}(L), \texttt{dl}(L)).$$

```
rdone : cert     rlist  :          list (int * int * int) -> cert
                 rlisti : int ->   list (int * int * int) -> cert
```

$$\forall R.\ f_c(\texttt{rlist}(R), \texttt{rlist}(R)).$$
$$\forall C \forall l \forall R.\ store_c(\texttt{rlisti}(l, R), C, \texttt{rlist}(R), \texttt{idx}(l)).$$
$$true_e(\texttt{rdone}).$$
$$\forall I \forall \Theta.\ decide_e(\texttt{rlist}([]), \Theta, \texttt{rdone}, \texttt{idx}(I))\ \texttt{:-}\ \langle \texttt{idx}(I), true \rangle \in \Theta.$$
$$\forall I, J, K, R, C, N, \Theta.\ cut_e(\texttt{rlist}([\langle I, J, K \rangle | R]), \Theta, \texttt{dl}([I, J]), \texttt{rlisti}(K, R), N)\ \texttt{:-}$$
$$\langle \texttt{idx}(K), C \rangle \in \Theta,\ \texttt{negate}(C, N).$$

Fig. 4. Resolution certificate definition in two parts

directly convert certain grammar specifications into parsers [29]. Logic programming languages such as Prolog, λProlog, and Elf have also been successfully used to provide direct implementations of the operational semantic specifications of programming languages [5, 21, 20].

Given that the kernel of an FPC checker is specified by inference rules (such as those in Figure 2) and that some forms of proof reconstruction should be supported during proof checking, a natural programming language for this reference checker is a logic programming language. The λProlog programming language [24] is a particularly good choice since it contains typing, abstract datatypes, and higher-order programming in a style similar to ML—the first programming language designed for implementing proof checkers [13]. λProlog goes beyond ML by providing a logically clean notion of binding and (object-level) substitution. Furthermore, λProlog implements both unification and backtracking search, two features critical for implementing proof reconstruction. These two features allow proof certificates to have the option of eliding some proof evidence in the hope that the proof checker can reconstruct the missing details. Allowing a trade-off between certificate size and checking (and proof reconstruction) time is an important feature for designing flexible proof certificate formats [8]. For example, this trade-off makes it possible for the `cnf` certificate format to not explicitly describe which literals are linked to which literals within a clause: without the

explicit information available, the logic programming implementation will do a simple and bounded search to find such linkable literals.

5 Related and future work

Dependent typed λ-calculi have been proposed as frameworks for specifying proof systems in a range of settings. The LF system [14] showed how natural deduction systems for intuitionistic logic could be given elegant and compact specifications. A logic programming language Elf [30] has also been built on top of LF: checking a proof in LF can then involve proving a goal in such a logic programming language. Unification and backtracking search are trusted components for Elf and partial proof reconstruction is possible in that setting.

The LF system (also called the $\lambda\Pi$-calculus) has been extended to allow for *deduction modulo* [9]: in the resulting system, implemented as Dedukti [33], functional computation replaces the proof search style computations within Elf. The LF system has also been extended with side conditions [32] and with external predicates [15] in order to make that proof representation more expressive.

Actually, the *LKF* based kernel that we have described in this paper is not the most expressive. We have experimented with writing kernels for both intuitionistic and classical logics; these are based on linear logic principles as described by the LKU focused proof system [19]. Using techniques developed by Chaudhuri [6], it should be possible to get a completely functioning *LKF* kernel from implementing just an *LJF* kernel.

Besides writing specifications of a number of other forms of proof evidence, we also plan to develop a similar approach to proof involving inductive and co-inductive definitions. With that extension, we should be able to check proof evidence coming from model checkers and inductive theorem provers. We are still doing the research to develop appropriate focusing systems for, essentially, classical and intuitionistic versions of arithmetic: the linear logic theory of fixed points developed by Baelde [4] is our current starting point.

Acknowledgments. The work presented in this paper has been done jointly with Zakaria Chihani and Fabien Renaud and has been funded by the ERC Advanced Grant ProofCert. I thank the anonymous reviewers 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(2):193–214, 1981.
3. Michaël Armand, Germain Faure, Benjamin Grégoire, Chantal Keller, Laurent Théry, and Benjamin Werner. A modular integration of SAT/SMT solvers to coq through proof witnesses. In J.-P. Jouannaud and Z. Shao, editors, *Certified Programs and Proofs (CPP 2011)*, *LNCS* 7086, pages 135–150, 2011.

4. David Baelde. Least and greatest fixed points in linear logic. *ACM Trans. on Computational Logic*, 13(1), April 2012.
5. P. Borras, D. Clément, Th. Despeyroux, J. Incerpi, G. Kahn, B. Lang, and V. Pascual. Centaur: the system. In *Third Annual Symposium on Software Development Environments (SDE3)*, pages 14–24, Boston, 1988.
6. Kaustuv Chaudhuri. Classical and intuitionistic subexponential logics are equally expressive. In Anuj Dawar and Helmut Veith, editors, *CSL 2010: Computer Science Logic*, *LNCS* 6247, pages 185–199, Brno, Czech Republic, August 2010. Springer.
7. 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.
8. Zakaria Chihani, Dale Miller, and Fabien Renaud. Foundational proof certificates in first-order logic. In Maria Paola Bonacina, editor, *CADE 24: Conference on Automated Deduction 2013*, number 7898 in LNAI, pages 162–177, 2013.
9. Denis Cousineau and Gilles Dowek. Embedding pure type systems in the lambda-pi-calculus modulo. In Simona Ronchi Della Rocca, editor, *Typed Lambda Calculi and Applications, 8th International Conference, TLCA 2007, Paris, France, June 26-28, 2007, Proceedings*, *LNCS* 4583, pages 102–117. Springer, 2007.
10. 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*, *LNCS* 3920, pages 167–181. Springer, 2006.
11. Gerhard Gentzen. Investigations into logical deduction. 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. Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
13. Michael J. Gordon, Arthur J. Milner, and Christopher P. Wadsworth. *Edinburgh LCF: A Mechanised Logic of Computation*, *LNCS* 78. Springer, 1979.
14. Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, 1993.
15. Furio Honsell, Marina Lenisa, Luigi Liquori, Petar Maksimovic, and Ivan Scagnetto. LFP: a logical framework with external predicates. In *LFMTP'12: Proceedings of the Seventh International Workshop on Logical Frameworks and Meta-languages, Theory and Practice*, pages 13–22. ACM New York, 2012.
16. Gilles Kahn. Natural semantics. In *Proceedings of the Symposium on Theoretical Aspects of Computer Science*, *LNCS* 247, pages 22–39. Springer, March 1987.
17. Chuck Liang and Dale Miller. Focusing and polarization in intuitionistic logic. In J. Duparc and T. A. Henzinger, editors, *CSL 2007: Computer Science Logic*, *LNCS* 4646, pages 451–465. Springer, 2007.
18. Chuck Liang and Dale Miller. Focusing and polarization in linear, intuitionistic, and classical logics. *Theoretical Computer Science*, 410(46):4747–4768, 2009.
19. Chuck Liang and Dale Miller. A focused approach to combining logics. *Annals of Pure and Applied Logic*, 162(9):679–697, 2011.
20. Spiro Michaylov and Frank Pfenning. Natural semantics and some of its meta-theory in Elf. In Lars Hallnäs, editor, *Extensions of Logic Programming*. Springer LNCS, 1992.
21. Dale Miller. Formalizing operational semantic specifications in logic. *Concurrency Column of the Bulletin of the EATCS*, October 2008.

22. Dale Miller. Proofcert: Broad spectrum proof certificates. An ERC Advanced Grant funded for the five years 2012-2016, February 2011.
23. Dale Miller. A proposal for broad spectrum proof certificates. In J.-P. Jouannaud and Z. Shao, editors, *CPP: First International Conference on Certified Programs and Proofs*, *LNCS* 7086, pages 54–69, 2011.
24. Dale Miller and Gopalan Nadathur. *Programming with Higher-Order Logic*. Cambridge University Press, June 2012.
25. 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.
26. Dale Miller, Gopalan Nadathur, and Andre Scedrov. Hereditary Harrop formulas and uniform proof systems. In David Gries, editor, *2nd Symp. on Logic in Computer Science*, pages 98–105, Ithaca, NY, June 1987.
27. Robin Milner, Mads Tofte, and Robert Harper. *The Definition of Standard ML*. MIT Press, 1990.
28. Lawrence C. Paulson and Jasmin Christian Blanchette. Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In *IWIL-LPAR*, pages 1–11, 2010.
29. Fernando C. N. Pereira and David H. D. Warren. Definite clauses for language analysis. *Artificial Intelligence*, 13:231–278, 1980.
30. Frank Pfenning. Elf: A language for logic definition and verified metaprogramming. In *4th Symp. on Logic in Computer Science*, pages 313–321, Monterey, CA, June 1989.
31. Gordon Plotkin. A structural approach to operational semantics. DAIMI FN-19, Aarhus University, Aarhus, Denmark, September 1981.
32. Aaron Stump. Proof checking technology for satisfiability modulo theories. In A. Abel and C. Urban, editors, *Logical Frameworks and Meta-Languages: Theory and Practice*, 2008.
33. The Dedukti team. The Dedukti system and homepage. `https://www.rocq.inria.fr/deducteam/Dedukti/index.html`, 2013.