A general proof certification framework for modal logic

Tomer Libal[1] and Marco Volpe[1]

Inria Saclay
{tomer.libal,marco.volpe}@inria.fr

Abstract

One of the main issues in proof certification is that different theorem provers, even when designed for the same logic, tend to use different proof formalisms and to produce outputs in different formats. The project ProofCert promotes the usage of a common specification language and of a small and trusted kernel in order to check proofs coming from different sources and for different logics. By relying on that idea and by using a classical focused sequent calculus as a kernel, we propose here a general framework for checking modal proofs. We present the implementation of the framework in a prolog-like language and show how it is possible to specialize it in a simple and modular way in order to cover different proof formalisms, such as labeled systems, tableaux, sequent calculi and nested sequent calculi. We illustrate the method for the logic K by providing several examples and discuss how to further extend the approach.

1 Introduction

One of the main issues in proof checking and proof certification is that proof evidences, even for a single, specific logic, are produced by using several different proof formalisms and proof calculi. This is the case both for human-generated proofs and for proofs provided by automated theorem provers, which moreover tend to produce outputs in different formats. Facing such an issue is one of the goals of the project ProofCert [21]. By using well-established concepts of proof theory, ProofCert proposes *foundational proof certificates* (FPC) as a framework to specify proof evidence formats. Describing a format in terms of an FPC allows software to check proofs in this format, much like a context-free grammar allows a parser to check the syntactical correctness of a program. The parser in this case would be a kernel: a small and trusted component that checks a proof evidence with respect to an FPC specification.

Checkers [8] is a generic proof certifier based on the ProofCert ideas. It allows for the certification of arbitrary proof evidences using various trusted kernels, like the focused classical sequent calculus *LKF* [18]. Such kernels are enriched with additional predicates, which allow for having more control on the construction of a proof. Dedicated FPC specifications can be defined, over these predicates, in order to interpret the information coming from a specific proof evidence format, so that the kernel is forced to produce a proof that mirrors, and thus certifies in case of success, the original one.

The problem of proliferation of proof formalisms and proof systems is especially apparent in the case of modal logics, whose proof theory is notoriously non-trivial. In fact, in the last decades several proposals have been provided (a general account is, e.g., in [10]). Such proposals range over a set of different proof formalisms (e.g., sequent, nested sequent, labeled sequent, hypersequent calculi, semantic tableaux), each of them presenting its own features and drawbacks. Several results concerning correspondences and connections between the different formalisms are also known [11, 13, 16].

In [20], a general framework for emulating and comparing existing modal proof systems has been presented. Such a framework is based on the setting of *labeled deduction systems* [12], which consists in enriching the syntax of modal logic with elements coming from the semantics, i.e., with elements referring explicitly to the worlds of a Kripke structure and to the accessibility relation between such worlds. In particular, the framework is designed as a focused version of Negri's system G3K [24], further enriched with a few parametric devices. Playing with such parameters produce concrete instantiations of the framework, which, by exploiting the expressiveness of the labeled approach and the control mechanisms of focusing, can be used to emulate the behavior of a range of existing formalisms and proof systems for modal logic with high precision.

In this paper, we rely on the close relationship between labeled sequent systems and *LKF* [23] in order to propose an implementation of such a framework that uses *LKF* as a kernel, and is developed as a module of the more general Checkers implementation project. This work also capitalizes on (and, in a sense, generalizes) the one in [19], which was limited to the case of prefixed tableaux and some of its variants. The implementation is extremely modular and based on the use of layers that mirror quite closely the instantiations of the framework presented in [20]. Concretely, we are able to certify, via this implementation, proofs given in the formalisms of labeled sequents, prefixed tableaux, ordinary sequent systems and nested sequents. We cover for the moment only the basic modal logic K, but the modularity of the approach should allow for an easy extension to other modal logics, in particular those whose Kripke frames are defined by geometric axioms, according to the treatment described in [20]. Extension to other formalisms seems also possible; we discuss this in more detail in the conclusion.

To the best of our knowledge, this project is the first attempt to independently certify the proofs generated by propositional modal proof systems.

We proceed as follows. In Section 2, we present some background on ProofCert, modal logic and proof systems for modal logic. In Section 3, we recall the general framework of [20]. In Section 4, we describe its implementation, by presenting the FPC specifications of the different layers and by providing a few examples. In Section 5, we conclude and discuss some possible future work.

2 Background

2.1 A general proof checker

There is no consensus about what shape should a formal proof evidence take. The notion of structural proofs, which is based on derivations in some calculus, is of no help as long as the calculus is not fixed. One of the ideas of the ProofCert project is to try to amend this problem by defining the notion of a foundational proof certificate (FPC) as a pair of an arbitrary proof evidence and an executable specification which denotes its semantics in terms of some well known target calculus, such as the Sequent Calculus. These two elements of an FPC are then given to a universal proof checker which, by the help of the FPC, is capable of deriving a proof in the target calculus. Since the proof generated is over a well known and low-level calculus which is easy to implement, one can obtain a high degree of trust in its correctness.

The proof certifier Checkers is a λProlog [22] implementation of this idea. Its main components are the following:

- **Kernel.** The kernels are the implementations of several trusted proof calculi. Currently, there are kernels over the classical and intuitionistic focused sequent calculus. Section 2.2 is devoted to present *LKF*, i.e. the classical focused sequent calculus that will be used in the paper.
- **Proof evidence.** The first component of an FPC, a proof evidence is a λProlog description of a proof output of a theorem prover. Given the high-level declarative form of λProlog,

the structure and form of the evidence are very similar to the original proof. We will see the precise form of the different proof evidences we handle in Section 4.

- **FPC specification.** The basic idea of Checkers is to try and generate a proof of the theorem of the evidence in the target kernel. In order to achieve that, the different kernels have additional predicates which take into account the information given in the evidence. Since the form of this information is not known to the kernel, Checkers uses FPC specifications in order to interpret it. These logical specifications are written in λProlog and interface with the kernel in a sound way in order to certify proofs. Writing these specifications is the main task for supporting the different outputs of the modal theorem provers we consider in this paper and they are, therefore, explained in detail in Section 4.

2.2 Classical Focused Sequent Calculus

Theorem provers usually employ efficient proof calculi with a lower degree of trust. At the same time, traditional proof calculi like the sequent calculus enjoy a high degree of trust but are very inefficient for proof search. In order to use the sequent calculus as the basis of automated deduction, much more structure within proofs needs to be established. Focused sequent calculi, first introduced by Andreoli [1] for linear logic, combine the higher degree of trust of sequent calculi with a more efficient proof search. They take advantage of the fact that some of the rules are "invertible", i.e. can be applied without requiring backtracking, and that some other rules can "focus" on the same formula for a batch of deduction steps. In this paper, we will make use of the classical focused sequent calculus (*LKF*) system defined in [18]. Fig. 1 presents, in the black font, the rules of *LKF*.

Formulas in *LKF* can have either positive or negative polarity and are constructed from atomic formulas, whose polarity has to be assigned, and from logical connectives whose polarity is pre-assigned. The connectives $\wedge^-, \vee^-$ and $\forall$ are of negative polarity, while $\wedge^+, \vee^+$ and $\exists$ are of positive polarity.

Deductions in *LKF* are done during invertible or focused phases. Invertible phases correspond to the application of invertible rules to negative formulas while a focused phase corresponds to the application of focused rules to a specific, focused, positive formula. Phases can be changed by the application of structural rules. A polarized formula A is a *bipolar formula* if A is a positive formula and no positive sub-formula occurrence of A is in the scope of a negative connective in A. A *bipole* is a pair of a negative phase below a positive phase within *LKF*: thus, bipoles are macro inference rules in which the conclusion and the premises are $\Uparrow$-sequents with no formulas to the right of the up-arrow.

It might be useful sometimes to delay the application of invertible rules (focused rules) on some negative formulas (positive formulas) A. In order to achieve that, we define the following delaying operators $\partial^+(A) = \texttt{true} \wedge^+ A$ and $\partial^-(A) = \texttt{false} \vee^- A$. Clearly, $A, \partial^+(A)$ and $\partial^-(A)$ are all logically equivalent but $\partial^+(A)$ is always considered as a positive formula and $\partial^-(A)$ as negative.

In order to integrate the use of FPC into the calculus, we enrich each rule of *LKF* with proof evidences and additional predicates, given in blue font in Fig. 1. We call the resulted calculus LKF^a. LKF^a extends *LKF* in the following way. Each sequent now contains additional information in the form of the proof evidence Ξ. At the same time, each rule is associated with a predicate (for example $initial_e(\Xi, l)$) which, according to the proof evidence, might prevent the rule from being called or guide it by supplying such information as the cut formula to be used.

Note that adding the FPC definitions in Fig. 1 does not harm the soundness of the system but only restricts the possible rules which can be applied at each step. Therefore, a proof

Invertible Rules

$$\frac{\Xi' \vdash \Theta \Uparrow A, \Gamma \quad \Xi'' \vdash \Theta \Uparrow B, \Gamma \quad \text{andNeg}_c(\Xi, \Xi', \Xi'')}{\Xi \vdash \Theta \Uparrow A \wedge^- B, \Gamma}$$

$$\frac{\Xi' \vdash \Theta \Uparrow A, B, \Gamma \quad \text{orNeg}_c(\Xi, \Xi')}{\Xi \vdash \Theta \Uparrow A \vee^- B, \Gamma} \qquad \frac{(\Xi' y) \vdash \Theta \Uparrow [y/x]B, \Gamma \quad \text{all}_c(\Xi, \Xi')}{\Xi \vdash \Theta \Uparrow \forall x.B, \Gamma}\ \dagger$$

Focused Rules

$$\frac{\Xi' \vdash \Theta \Downarrow B_1 \quad \Xi'' \vdash \Theta \Downarrow B_2 \quad \text{andPos}_e(\Xi, \Xi', \Xi'')}{\Xi \vdash \Theta \Downarrow B_1 \wedge^+ B_2}$$

$$\frac{\Xi' \vdash \Theta \Downarrow B_i \quad \text{orPos}_e(\Xi, \Xi', i)}{\Xi \vdash \Theta \Downarrow B_1 \vee^+ B_2} \qquad \frac{\Xi' \vdash \Theta \Downarrow [t/x]B \quad \text{some}_e(\Xi, t, \Xi')}{\Xi \vdash \Theta \Downarrow \exists x.B}$$

Identity rules

$$\frac{\Xi' \vdash \Theta \Uparrow B \quad \Xi'' \vdash \Theta \Uparrow \neg B \quad cut_e(\Xi, \Xi', \Xi'', B)}{\Xi \vdash \Theta \Uparrow \cdot}\ cut \qquad \frac{\langle l, \neg P_a \rangle \in \Theta \quad initial_e(\Xi, l)}{\Xi \vdash \Theta \Downarrow P_a}\ init$$

Structural rules

$$\frac{\Xi' \vdash \Theta \Uparrow N \quad release_e(\Xi, \Xi')}{\Xi \vdash \Theta \Downarrow N}\ release \qquad \frac{\Xi' \vdash \Theta, \langle l, C \rangle \Uparrow \Gamma \quad store_c(\Xi, C, l, \Xi')}{\Xi \vdash \Theta \Uparrow C, \Gamma}\ store$$

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

Figure 1: The augmented *LKF* proof system LKF^a. The proviso † requires that y is not free in Ξ, Θ, Γ, B. The symbol P_a denotes a positive atomic formula.

obtained using LKF^a is also a proof in *LKF*. Since the additional predicates do not compromise the soundness of LKF^a, we allow their definition to be external to the kernel and in fact these definitions, which are supplied by the user, are what allow Checkers to check arbitrary proof formats. Section 4 is mainly devoted to the definitions of these programs for the different proof formats of the modal theorem provers.

2.3 Proof systems for modal logic

In this section, we review several proof systems that are among the most popular calculi for automated theorem proving in modal logic as well as for manual proof generation. Before that, we recall a few key notions about modal logic and its relation with first-order classical logic.

2.3.1 Modal logic

The language of *(propositional) modal formulas* consists of a functionally complete set of classical propositional connectives, a *modal operator* $\Box$ (here we will also use explicitly its dual $\Diamond$) and a denumerable set $\mathcal{P}$ of *propositional symbols*. Along this paper, we will work with formulas in *negation normal form*, i.e., such that only atoms may possibly occur negated in them. Notice that this is not a restriction, as it is always possible to convert a propositional modal formula into an equivalent formula in negation normal form. The grammar is specified as follows:

$$A ::= P \mid \neg P \mid A \vee A \mid A \wedge A \mid \Box A \mid \Diamond A$$

where $P \in \mathcal{P}$. We say that a formula is a $\Box$-*formula* ($\Diamond$-*formula*) if its main connective is $\Box$ ($\Diamond$). The semantics of the modal logic K is usually defined by means of *Kripke frames*, i.e., pairs $\mathcal{F} = (W, R)$ where W is a non empty set of *worlds* and R is a binary relation on W. A *Kripke model* is a triple $\mathcal{M} = (W, R, V)$ where (W, R) is a Kripke frame and $V : W \rightarrow 2^{\mathcal{P}}$ is a function that assigns to each world in W a (possibly empty) set of propositional symbols.

In the basic modal logic K, we define the *truth* of a modal formula at a point w in a Kripke structure $\mathcal{M} = (W, R, V)$ as the smallest relation $\models$ satisfying:

$$\begin{array}{rcl}
\mathcal{M}, w \models P & \text{iff} & P \in V(w) \\
\mathcal{M}, w \models \neg P & \text{iff} & P \notin V(w) \\
\mathcal{M}, w \models A \vee B & \text{iff} & \mathcal{M}, w \models A \text{ or } \mathcal{M}, w \models B \\
\mathcal{M}, w \models A \wedge B & \text{iff} & \mathcal{M}, w \models A \text{ and } \mathcal{M}, w \models B \\
\mathcal{M}, w \models \Box A & \text{iff} & \mathcal{M}, w' \models A \text{ for all } w' \text{ s.t. } wRw' \\
\mathcal{M}, w \models \Diamond A & \text{iff} & \text{there exists } w' \text{ s.t. } wRw' \text{ and } \mathcal{M}, w' \models A.
\end{array}$$

By extension, we write $\mathcal{M} \models A$ when $\mathcal{M}, w \models A$ for all $w \in W$ and we write $\models A$ when $\mathcal{M} \models A$ for every Kripke structure $\mathcal{M}$.

2.3.2 The standard translation from modal logic into classical logic

The following *standard translation* (see, e.g., [4]) provides a bridge between propositional (classical) modal logic and first-order classical logic:

$$\begin{array}{rcl@{\qquad}rcl}
ST_x(P) & = & P(x) & ST_x(A \wedge B) & = & ST_x(A) \wedge ST_x(B) \\
ST_x(\neg P) & = & \neg P(x) & ST_x(\Box A) & = & \forall y(R(x,y) \supset ST_y(A)) \\
ST_x(A \vee B) & = & ST_x(A) \vee ST_x(B) & ST_x(\Diamond A) & = & \exists y(R(x,y) \wedge ST_y(A))
\end{array}$$

where x is a free variable denoting the world in which the formula is being evaluated. The first-order language into which modal formulas are translated is usually referred to as *first-order correspondence language* [4] and consists of a binary predicate symbol R and a unary predicate symbol P for each $P \in \mathcal{P}$. When a modal operator is translated, a new fresh variable is introduced. It is easy to show that for any modal formula A, any model $\mathcal{M}$ and any world w, we have that $\mathcal{M}, w \models A$ if and only if $\mathcal{M} \models ST_x(A)[x \leftarrow w]$.

2.3.3 Labeled sequent systems

Several different deductive formalisms have been used for modal proof theory and theorem proving. One of the most interesting approaches has been presented in [12] with the name of labeled deduction. The basic idea behind labeled proof systems for modal logic is to internalize elements of the corresponding Kripke semantics (namely, the worlds of a Kripke structure and the accessibility relation between such worlds) into the syntax. A concrete example of such a system is the sequent calculus $G3K$ presented in [24] (we will refer to it as *LS* in this paper). *LS formulas* are either *labeled formulas* of the form $x : A$ or *relational atoms* of the form xRy, where x, y range over a set of variables and A is a modal formula. In the following, we will use φ, ψ to denote *LS* formulas. *LS sequents* have the form $\Gamma \vdash \Delta$, where Γ and Δ are multisets containing labeled formulas and relational atoms. In Fig. 2, we present the rules of *LS*, which is proved to be sound and complete for the basic modal logic K [24].

Classical rules

$$\frac{}{x : P, \Gamma \vdash \Delta, x : P}\ init \qquad \frac{x : A, x : B, \Gamma \vdash \Delta}{x : A \wedge B, \Gamma \vdash \Delta}\ L\wedge \quad \frac{\Gamma \vdash \Delta, x : A \quad \Gamma \vdash \Delta, x : B}{\Gamma \vdash \Delta, x : A \wedge B}\ R\wedge$$

$$\frac{x : A, \Gamma \vdash \Delta \quad x : B, \Gamma \vdash \Delta}{x : A \vee B, \Gamma \vdash \Delta}\ L\vee \quad \frac{\Gamma \vdash \Delta, x : A, x : B}{\Gamma \vdash \Delta, x : A \vee B}\ R\vee$$

Modal rules

$$\frac{y : A, x : \Box A, xRy, \Gamma \vdash \Delta}{x : \Box A, xRy, \Gamma \vdash \Delta}\ L\Box \quad \frac{xRy, \Gamma \vdash \Delta, y : A}{\Gamma \vdash \Delta, x : \Box A}\ R\Box \quad \frac{xRy, y : A, \Gamma \vdash \Delta}{x : \Diamond A, \Gamma \vdash \Delta}\ L\Diamond \quad \frac{xRy, \Gamma \vdash \Delta, x : \Diamond A, y : A}{xRy, \Gamma \vdash \Delta, x : \Diamond A}\ R\Diamond$$

In $R\Box$ and $L\Diamond$, y does not occur in the conclusion.

Figure 2: *LS*: a labeled sequent system for the modal logic K

Classical rules

$$\frac{\sigma : A \wedge B}{\sigma : A,\ \sigma : B}\ \wedge_F \qquad \frac{\sigma : A \vee B}{\sigma : A \quad | \quad \sigma : B}\ \vee_F$$

Modal rules

$$\frac{\sigma : \Box A}{\sigma.n : A}\ \Box_F \qquad \frac{\sigma : \Diamond A}{\sigma.n : A}\ \Diamond_F$$

In $\Box_F$, $\sigma.n$ is used. In $\Diamond_F$, $\sigma.n$ is new.

Figure 3: *PT*: a prefixed tableau system for the modal logic K

2.3.4 Prefixed tableau systems

Prefixed tableaux (*PT*) can also be seen as a particular kind of labeled deductive system. They were introduced in [9]. The formulation that we use here is closer to the one in [10] and it is given in terms of unsigned formulas. A *prefix* is a finite sequence of positive integers (written by using dots as separators). Intuitively, prefixes denote possible worlds and they are such that if σ is a prefix, then $\sigma.1$ and $\sigma.2$ denote two worlds accessible from σ. A *prefixed formula* is $\sigma : A$, where σ is a prefix and A is a modal formula in negation normal form. A prefixed tableau proof of A starts with a root node containing $1 : A$, informally asserting that A is false in the world named by the prefix 1. It continues by using the branch extension rules given in Figure 3. We say that a branch of a tableau is a *closed branch* if it contains $\sigma : P$ and $\sigma : \neg P$ for some σ and some P. The goal is to produce a *closed tableau*, i.e., a tableau such that all its branches are closed. Classical rules in Figure 3 are the prefixed version of the standard ones. For what concerns the modal rules, the $\Diamond$ rule applied to a formula $\sigma : A$ intuitively allows for generating a new world, accessible from σ, where A holds, while the $\Box$ rule applied to a formula $\Box : A$ allows for moving the formula A to an already existing world accessible from σ. We say that a prefix is *used* on a branch if it already occurs in the tableau branch and it is *new* otherwise.

2.3.5 Ordinary sequent systems

Several "ordinary" sequent systems have been proposed in the literature for different modal logics (a general account is, e.g., in [14, 25]). In our treatment, we will use the formalization *OS* presented in Figure 4, which is adapted mainly from the presentations in [10, 26]. The base classical system (consisting of *identity*, *structural* and *classical connective* rules) is extended by a modal rule that works on one $\Box$-formula and several $\Diamond$-formulas.

Classical rules

$$\frac{}{\vdash \Gamma, P, \neg P}\ init \qquad \frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \wedge B}\ \wedge \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \vee B}\ \vee$$

Modal rules

$$\frac{\vdash \Gamma, A}{\vdash \Diamond\Gamma, \Box A, \Delta}\ \Box_K$$

Figure 4: *OS*: an ordinary sequent system for the modal logic K.

Classical rules

$$\frac{}{\mathcal{N}\{P, \neg P\}}\ init \qquad \frac{\mathcal{N}\{A\} \quad \mathcal{N}\{B\}}{\mathcal{N}\{A \wedge B\}}\ \wedge \qquad \frac{\mathcal{N}\{A, B\}}{\mathcal{N}\{A \vee B\}}\ \vee$$

Modal rules

$$\frac{\mathcal{N}\{[A]\}}{\mathcal{N}\{\Box A\}}\ \Box \qquad \frac{\mathcal{N}\{\Diamond A, [A, \mathcal{M}]\}}{\mathcal{N}\{\Diamond A, [\mathcal{M}]\}}\ \Diamond$$

Figure 5: *NS*: a nested sequent system for the modal logic K.

2.3.6 Nested sequent systems

Nested sequents (first introduced by Kashima [15], and then independently rediscovered by Poggiolesi [25], as *tree-hypersequents*, and by Brünnler [5]) are an extension of ordinary sequents to a structure of tree, where each [ ]-node represents the scope of a modal $\Box$. We write a nested sequent as a multiset of formulas and *boxed sequents*, according to the following grammar, where A can be any modal formula in negative normal form: $\mathcal{N} ::= \emptyset \mid A, \mathcal{N} \mid [\mathcal{N}], \mathcal{N}$

In a nested sequent calculus, a rule can be applied at any depth in this tree structure, that is, inside a certain nested sequent context. A *context* written as $\mathcal{N}\{\ \}\cdots\{\ \}$ is a nested sequent with a number of holes occurring in place of formulas (and never inside a formula). Given a context $\mathcal{N}\{\ \}\cdots\{\ \}$ with n holes, and n nested sequents $\mathcal{M}_1, \ldots, \mathcal{M}_n$, we write $\mathcal{N}\{\mathcal{M}_1\}\cdots\{\mathcal{M}_n\}$ to denote the nested sequent where the i-th hole in the context has been replaced by $\mathcal{M}_i$, with the understanding that if $\mathcal{M}_i = \emptyset$ then the hole is simply removed. We are going to consider the nested sequent system (on Figure 5) introduced by Brünnler in [5], that we call here *NS*.

3 A general focused framework for modal logic

3.1 A translation from the modal language into a first-order polarized language

In [23], it has been shown how it is possible to translate a modal formula A into a polarized first-order formula A' in such a way that a strict correspondence between rule applications in a *LS* proof of A and bipoles in an *LKF* proof of A' holds. Such a correspondence has been used in order to prove some adequacy theorem and to define a focused version of *LS*. Here we will further exploit it for checking labeled sequent and prefixed tableaux derivations in the augmented variant LKF^a.

The translation is obtained from the standard translation of Section 2.3.2 by adding some elements of polarization. First of all, when translating a modal formula into a polarized one, we are often in a situation where we are interested in putting a delay in front of the formula only in the case when it is negative and not a literal. For that purpose, we define A^{∂^+}, where A is a modal formula in negation normal form, to be A if A is a literal or a positive formula and $\partial^+(A)$ otherwise.

Given a world x, we define the translation $[.]_x$ from modal formulas in negation normal form into polarized first-order formulas as:

$$
\begin{array}{rclrcl}
[P]_x & = & P(x) & [A \wedge B]_x & = & {[A]_x}^{\partial^+} \wedge^- {[B]_x}^{\partial^+} \\
[\neg P]_x & = & \neg P(x) & [A \vee B]_x & = & {[A]_x}^{\partial^+} \vee^- {[B]_x}^{\partial^+} \\
[\Box A]_x & = & \forall y(\neg R(x,y) \vee^- {[A]_y}^{\partial^+}) & [\Diamond A]_x & = & \exists y(R(x,y) \wedge^+ \partial^-({[A]_y}^{\partial^+}))
\end{array}
$$

In this translation, delays are used to ensure that only one connective is processed along a given bipole, e.g., when we decide on (the translation of) a $\Diamond$-formula $[\Diamond A]_x$, the (translation of the) formula A is delayed in such a way that it gets necessarily stored at the end of the bipole. Based on that, we define the translation $[.]$ from labeled formulas and relational atoms into polarized first-order formulas as $[x : A] = [A]_x$ and $[xRy] = R(x, y)$. We will sometimes use the extension of this notion to multisets of labeled formulas, i.e., $[\Gamma] = \{[\varphi] \mid \varphi \in \Gamma\}$. Note that predicates of the form $P(x)$ and $R(x, y)$ are considered as having positive polarity. Finally, we define a translation from *LS* sequents into *LKF* sequents:

$$
[(\varphi_1, \dots, \varphi_n \vdash \psi_1, \dots, \psi_m)] = \vdash [\neg\varphi_1]^{\partial^+}, \dots, [\neg\varphi_n]^{\partial^+}, [\psi_1]^{\partial^+}, \dots, [\psi_m]^{\partial^+} \Uparrow \cdot
$$

where $[\neg\varphi]$ is $[(\neg A)]_x$ if $\varphi = x : A$ and is $\neg R(x, y)$ if $\varphi = xRy$.

We recall here a result from [23], where a more formal statement and a detailed proof can be found.

Theorem 1. *Let Π be a LS derivation of a sequent S from the sequents $S_1, \dots, S_n$. Then there exists an LKF derivation Π' of $[S]$ from $[S_1], \dots, [S_n]$, such that each rule application in Π corresponds to a bipole in Π'. The viceversa, for first-order formulas that are translation of modal formulas, also holds.*

3.2 A focused labeled framework

Theorem 1 ensures that we can easily check an *LS* proof by using a kernel based on *LKF* and the translation of Section 3.1. Given the tight correspondence between *LS* inference rules and *LKF* bipoles, the information concerning the original proof that we need in order to reproduce it faithfully [19] (typically going from the root to the leaves) in *LKF* is restricted to the following:

- at each step, the formula on which a rule is applied;
- when a $\Diamond$ rule is applied, which term is used as a witness;
- in the case of an initial rule, with respect to which pair of complementary literals it is applied.

Theorem 1 also led, in [23], to the definition of a focused labeled sequent system (*LMF*) for modal logic, which can be seen either as a focused version of Negri's system or as the restriction of *LKF* to the first-order correspondence language (where modalities are seen as synthetic connectives).

In the context of modal logics, labeled proof systems have been shown to be quite expressive and encodings of other approaches into this formalism have also been presented in the literature [11, 13, 16]. It seems therefore quite natural to explore the possibility of reproducing the behavior of modal proof systems based on different formalisms inside *LMF*, by exploiting at the same time the expressivity of labeling and the control mechanisms provided by focusing. Such an analysis has been carried out in [20] and has shown that, by enriching *LMF* with a few further technical devices, it is possible to get enough power to drive construction of proofs so as to emulate the proof structure of a wide range of formalisms. We illustrate the need for such further devices by considering the example of ordinary sequent systems, which present features also typical of other, related approaches. Consider the following typical sequent calculus rule for modal logic: [1]

$$\frac{\vdash\ \Gamma, A}{\vdash \Diamond\Gamma, \Box A}\ .$$

We observe that:

(i) this rule works at the same time on one $\Box$-formula and on n $\Diamond$-formulas. In order to process such $\Diamond$-formulas, in our labeled deduction setting, it is necessary to apply the $\Diamond$-introduction rule n times. Since these applications do not interfere with each other, they can, in fact, be applied in parallel. For this reason, we move to a multifocused [6] version of *LMF*, i.e., a variant where we can focus on several positive formulas at the same time. In this way, we can group all the $\Diamond$-introductions inside a single phase (in the following, we will sometimes call it a $\Diamond$-phase).

(ii) intuitively, one can read this inference rule (reading from conclusion to premise) as moving from one world to another (reachable) world in a suitable Kripke structure. Such a change of world becomes apparent when we consider the corresponding deduction steps in a labeled system, as, in this case, modal introduction rules will explicitly change the label of the formulas under consideration. In order to properly mimic the behavior of the original rule, in the labeled system we need to be able to force all the formulas involved in the rule to move to the same new world. We therefore modify the notion of a labeled formula to have the form $xy : A$. Here x indicates in which world such a formula holds, while y gets initialized when one multifocuses on the multiset of $\Diamond$-formulas and is used to drive future applications of $\Diamond$-rules. E.g., if $x : \Diamond\Gamma$ is on the left of $\Uparrow$, then we can multifocus on $xy : \Diamond\Gamma$ for a given y reachable from x. This y will be used as a witness in the application of a (properly modified) $\Diamond$-introduction rule, in such a way that at the end of the bipole, we will have the multiset $y : \Gamma$ on the left of $\Uparrow$.

(iii) in *LMF*, when constructing a proof tree (going from the root towards the leaves), formulas we decide on are duplicated and stay in the storage (that is, on the left of $\Uparrow$ or $\Downarrow$). It follows that all along a proof, it is possible to switch freely from one label to another in the deduction process. On the contrary, in a sequent calculus rule like the one given above, only formulas having a modal operator as the main connective can be "promoted" to a different world. According to the Kripke-style interpretation presented, this amounts to considering a single world at a time, in such a way that when moving to a new one, formulas standing at previously encountered worlds are not accessible anymore. In order to emulate this aspect, labelled sequents are further decorated with a set $\mathcal{H}$ of labels, specifying which worlds are currently enabled, with the intended meaning that we can decide on a formula only if its label belongs to $\mathcal{H}$.

The general framework LMF_* is presented in Figure 6[2].

[1] In Figure 4, we presented a variant of this rule where a context is also added in the conclusion. This simplified version is enough for the illustration purposes of this paragraph.

[2] Please note that the framework presented here is slightly different from the one in [20], since considering only the logic K allows for some simplification.

Asynchronous introduction rules

$$\frac{}{\mathcal{G} \vdash_{\mathcal{H}} \Theta \Uparrow x : t^-, \Omega}\; t^-{}_F \qquad \frac{\mathcal{G} \vdash_{\mathcal{H}} \Theta \Uparrow \Omega}{\mathcal{G} \vdash_{\mathcal{H}} \Theta \Uparrow x : f^-, \Omega}\; f^-{}_F$$

$$\frac{\mathcal{G} \vdash_{\mathcal{H}} \Theta \Uparrow x : A, \Omega \quad \mathcal{G} \vdash_{\mathcal{H}} \Theta \Uparrow x : B, \Omega}{\mathcal{G} \vdash_{\mathcal{H}} \Theta \Uparrow x : A \wedge^- B, \Omega}\; \wedge^-_F \qquad \frac{\mathcal{G} \vdash_{\mathcal{H}} \Theta \Uparrow x : A, x : B, \Omega}{\mathcal{G} \vdash_{\mathcal{H}} \Theta \Uparrow x : A \vee^- B, \Omega}\; \vee^-_F$$

$$\frac{\mathcal{G} \cup \{xRy\} \vdash_{\mathcal{H}} \Theta \Uparrow y : B, \Omega}{\mathcal{G} \vdash_{\mathcal{H}} \Theta \Uparrow x : \Box B, \Omega}\; \Box_F$$

Synchronous introduction rules

$$\frac{}{\mathcal{G} \vdash_{\mathcal{H}} \Theta \Downarrow x\sigma : t^+}\; t^+{}_F \qquad \frac{\mathcal{G} \vdash_{\mathcal{H}} \Theta \Downarrow x\sigma : B_1, \Omega_1 \quad \mathcal{G} \vdash_{\mathcal{H}} \Theta \Downarrow x\sigma : B_2, \Omega_2}{\mathcal{G} \vdash_{\mathcal{H}} \Theta \Downarrow x\sigma : B_1 \wedge^+ B_2, \Omega_1, \Omega_2}\; \wedge^+_F$$

$$\frac{\mathcal{G} \vdash_{\mathcal{H}} \Theta \Downarrow x\sigma : B_i, \Omega}{\mathcal{G} \vdash_{\mathcal{H}} \Theta \Downarrow x\sigma : B_1 \vee^+ B_2, \Omega}\; \vee^+_F, i \in \{1,2\} \qquad \frac{\mathcal{G} \cup \{xRy\} \vdash_{\mathcal{H}} \Theta \Downarrow y : B, \Omega}{\mathcal{G} \cup \{xRy\} \vdash_{\mathcal{H}} \Theta \Downarrow xy : \Diamond B, \Omega}\; \Diamond_F$$

Identity rules

$$\frac{}{\mathcal{G} \vdash_{\mathcal{H}} x : \neg B, \Theta \Downarrow x : B}\; init_F$$

Structural rules

$$\frac{\mathcal{G} \vdash_{\mathcal{H}} \Theta, x : B \Uparrow \Omega}{\mathcal{G} \vdash_{\mathcal{H}} \Theta \Uparrow x : B, \Omega}\; store_F \qquad \frac{\mathcal{G} \vdash_{\mathcal{H}} \Theta \Uparrow \Omega'}{\mathcal{G} \vdash_{\mathcal{H}} \Theta \Downarrow \Omega}\; release_F \qquad \frac{\mathcal{G} \vdash_{\mathcal{H}'} \Theta \Downarrow \Omega}{\mathcal{G} \vdash_{\mathcal{H}} \Theta \Uparrow \cdot}\; decide_F$$

In $store_F$, B is a positive formula or a negative literal.
In $init_F$, B is a positive literal.
In $\Box_F$, y is different from x and does not occur in $\mathcal{G}$ nor in Θ.
In $decide_F$, if $xy : A \in \Omega$ then $x : A \in \Theta$. Moreover, Ω contains only positive formulas of the form: (i) $x\sigma : A$, where A is not a $\Diamond$-formula and $x \in \mathcal{H}$; or (ii) $xy : A$ where A is a $\Diamond$-formula, $xRy \in \mathcal{G}$, $x \in \mathcal{H}$.
In $release_F$, Ω contains no positive formulas and $\Omega' = \{x : A \mid x\sigma : A \in \Omega\}$.

Figure 6: LMF_*: a focused labeled framework for the modal logic K.

In the rest of this paper, when talking of LMF_* and its instantiations, a *labeled formula* will have the form $\varphi \equiv x\sigma : A$, where σ is either empty or a label y. We say that x is the *present* of φ and σ is the *future* of φ. An LMF_* *sequent* has the form $\mathcal{G} \vdash_{\mathcal{H}} \Theta \Uparrow \Omega$ or $\mathcal{G} \vdash_{\mathcal{H}} \Theta \Downarrow \Omega$, where the *relational set (of the sequent)* $\mathcal{G}$ is a set of relational atoms, the *present (of the sequent)* $\mathcal{H}$ is a non-empty multiset of labels, and Θ and Ω are multisets of labeled formulas.

3.3 Emulation of modal proof systems

In order to emulate proofs given in other proof calculi by means of the focused framework LMF_*, we need to give a specialized version of the rule $decide_F$.

In order to define a translation $\lfloor\cdot\rfloor$ from modal formulas in negation normal form into polarized modal formulas, we refine the one given in Section 3.1, by considering the fact that we have now modal operators in the target language and do not need to translate explicitly modalities into quantifiers:

$$\begin{array}{rclcrcl}
\lfloor P \rfloor & = & P & \qquad & \lfloor A \wedge B \rfloor & = & \lfloor A \rfloor^{\partial^+} \wedge^- \lfloor B \rfloor^{\partial^+} \\
\lfloor \neg P \rfloor & = & \neg P & & \lfloor A \vee B \rfloor & = & \lfloor A \rfloor^{\partial^+} \vee^- \lfloor B \rfloor^{\partial^+} \\
\lfloor \Box A \rfloor & = & \Box(\lfloor A \rfloor^{\partial^+}) & & \lfloor \Diamond A \rfloor & = & \Diamond(\partial^-(\lfloor A \rfloor^{\partial^+}))
\end{array}$$

For *LS*, we specialize the rule $decide_F$ as follows:

$$\frac{\mathcal{G} \vdash_{\mathcal{L}} \Theta \Downarrow x\sigma : A}{\mathcal{G} \vdash_{\mathcal{L}} \Theta \Uparrow \cdot} \; decide_{LS}$$

where:

- $\mathcal{L}$ denotes the set of all labels;
- if A is a $\Diamond$ formula, then σ is y for some $xRy \in \mathcal{G}$; otherwise, σ is empty.

Given the similar nature of the approaches, in the case of the logic K, the same rule can be used also for emulating the systems *PT* and *NS* (for convenience, in the following we will use for the same rule also the names $decide_{PT}$ and $decide_{NS}$). Differences will emerge when considering logics beyond K as, e.g., the treatment of a rule for $S4$ in systems based on prefixed tableaux and nested sequents tend to use a principle similar to that applied in OS and consisting in moving a $\Box$ from a world to another reachable one. We also remark that the difference of approach between *LS* and *PT* is captured by a different translation of the original formula to be proved (which needs to be negated in the case of tableaux) rather than by differences in the decide rule.

For ordinary sequents, we specialize instead the rule $decide_F$ as follows:

$$\frac{\mathcal{G} \vdash_{\{y\}} \Theta \Downarrow \Omega}{\mathcal{G} \vdash_{\{x\}} \Theta \Uparrow \cdot} \; decide_{OS}$$

where (in addition to the general conditions of Figure 6) we have that:

1. if $x \neq y$, then:
   - $xRy \in \mathcal{G}$; and
   - Ω is a multiset of formulas of the form $xy : \Diamond A$;
2. if $x = y$, then $\Omega = \{x : A\}$ for some formula A that is not a $\Diamond$-formula.

Intuitively, the specialization with respect to the general framework consists in: (i) restricting the use of multifocusing to $\Diamond$-formulas; (ii) forcing such $\Diamond$-formulas to be labeled with the same future.

Let X range over $\{LS, PT, OS, NS\}$. We call LMF_X the system obtained from LMF_* by replacing the rule $decide_F$ with the rule $decide_X$. The adequacy of the implementation proposed in next section relies on the following result, which is proved by associating to each rule in X a *corresponding* sequence of bipoles in LMF_X. We refer the reader to [20] for a more formal statement of the theorem as well as for its complete proof.

Theorem 2. *Let X range over $\{LS, PT, OS, NS\}$. There exists a proof Π of A in the proof system X iff there exists a proof Π' of $\emptyset \vdash_{\{x\}} x : (\lfloor A \rfloor)^{\partial^+} \Uparrow \cdot$, for some x, in LMF_X. Moreover, for each application of a rule r in Π there is a sequence of bipoles in Π' corresponding to r.*

4 Certification of modal proofs

This section describes the implementation of a general framework for the certification of modal proofs and shows how this framework can be used in order to certify proofs from different proof systems. We will rely here on the theoretical results of Section 3. We just notice that in practice we do not use the labeled modal system LMF_* as a kernel but rather implement it on top of LKF^a. This allows for keeping a simple and uniform kernel in the context of the Checkers project, that also considers other logics and formalisms. However, given Theorem 1, the adequacy result of Theorem 2 automatically transfers from LMF_X to LKF.

4.1 A proof certification framework

Foundational proof certificates form a rich language for the certification of any proof object. This richness has the downside that defining a new set of FPC specifications is, in general, a complex task. This property is not unique to ProofCert. There are but a few general proof certification tools and the effort to enable the certification of a particular proof system is non-trivial.

Our aim in this paper is to enable both generalization and ease of use. This is going to be attempted by the development of a *layered* framework, where each layer is defined in terms of the previous one. This framework is an implementation of the systems described in section 3.2, where each layer in our implementation directly corresponds to one of the systems described there. Moreover, we take the incremental build-up of systems in the paper one step further and implement each framework in our system in terms of the previous one. Such a layered framework will restrict the richness of the foundational proof certificates in a way that will make it easier to develop FPC specifications which can be used to efficiently certify various other systems. To preserve the generality of the system for modal logic, the top layer will be capable of certifying arbitrary other systems. The bottom level of this framework will be the *LMF* system, which is similar to the one described in [19]. This system will be extended to a simulation of multi-focusing and will result in the system LMF^m. The final system is LMF^*.

The definition of each layer is characterized by three elements:

1. A supported proof format
2. Its FPC specification
3. A monad-like state

In order to support our layered architecture, we had to use techniques such as abstraction, encapsulation, polymorphism and modularity. Such techniques are not native to logic programming languages and were simulated in our system by the combination of a careful accumulation of files, using constants to move between layers, a set of "conversion" function and λProlog types.

It should be noted that the state is not an integral part of the proof evidence. The fact that we include it in the certificate is done only in order to simplify the implementation. The state of each layer is being initialized by fixed constants and can be, therefore, omitted from the evidence. We will describe the state in some details when speaking about the frameworks but omit such discussion when describing the supported proof evidence.

4.1.1 The *LMF* system layer

In [19] we have presented a system which is capable of certifying several labeled sequent and prefixed tableau based proof systems.

We have shown, that given the correspondence between rule applications in the original calculus and bipoles in *LKF*, we can state an easy and faithful encoding of proofs, mainly based on specifying on which formulas we decide every time we start a new bipole.

Our first layer is capable, therefore, of accepting proof evidence which contains the following information:

1. at each step, on which formula we apply a rule;
2. in the case of a $\Diamond$-formula, with respect to which $\Box$-formula we apply the rule;
3. in the case of an initial rule, with respect to which complementary literal we apply it.

For this reason, we define the proof evidence of this layer to consist in a tree describing the original proof. Each node is decorated by a pair containing: (i) the formula on which a rule is applied, as explained in (1), together with (ii) a (possibly null) further index carrying additional information, to be used in cases (2) and (3) above. Formulas in the tree will drive the construction (bottom-up) of the *LKF* derivation, in the sense that, by starting from the root, at each step, the *LKF* kernel will decide on the given formula and proceed, constrained by properly defined clerks and experts, along a positive and a negative phase. The results in [19] guarantees that at the end of a bipole, we will be in a situation which is equivalent to that of the corresponding step in the original proof.

As described in item (2) above, if we are applying an $\exists$-rule in *LKF*, then we need further information specifying with respect to which eigenvariable we apply the rule. This is done by linking, using the state, the formula under consideration to the corresponding new-world-generating $\Box$-formula. Similarly, in the case of an initial (3), the additional information in the node will specify the index of the complementary literal. This information will be captured in a state-monad which will capture, in this layer and in the following ones, all information which is independent of the evidence but is required for the correct execution of the system.

In order to provide an FPC specification for a particular format, we need to define the specific items that are used to augment *LKF*. In particular, the constructors for proof certificate terms and for indices must be provided: this is done in λ-Prolog by declaring constructors of the types `cert` and `index`.

The indexing mechanism is defined next.

```
% defined in lmf-singlefoc.sig
type root index.
type lind index -> index.
type rind index -> index.
type diaind index -> index -> index.
type none index.
```

The `lind` and `rind` indices are functions denoting the left and right sub-formulas. The `root` index is a constant denoting the root formula. In order to simulate the different labels associated with different applications of the same $\Diamond$-formula, we are using the `diaind` function which also refers to the associated box. The `none` index just allows us to denote indices as optional data structures.

Figure 7 gives an example of the relationship between indices and sub-formulas. As mentioned above, since the same $\Diamond$-formula can be associated with different $\Box$-formulas, we use a specific index, the `diaind` for its sub-formula.

In order to be able to transform the same proof object between different layers, we have defined a notion of abstract tree as follows:

```
% defined in lmf-singlefoc.sig
kind lmf-node, lmf-tree type.
type lmf-tree lmf-node -> list lmf-tree -> lmf-tree.
```

```
root -> ((□p) ∧ (◇¬q)) ∨ (□(¬p ∨ q))            (lind root) -> (□p) ∧ (◇¬q)
(lind (lind root)) -> □p                        (rind (lind root)) -> ◇¬q
(diaind (rind (lind root)) (rind root)) -> ¬q
```

Figure 7: Possible indexing of sub-formulas of $((\Box p) \wedge (\Diamond \neg q)) \vee (\Box(\neg p \vee q))$

This definition permits the usage of different types of nodes in the same tree, which will allow us to smoothly move between the layers.

Using these definitions, we can now give the definition of the supported proof format.

```
% defined in lmf-singlefoc.sig
kind lmf-singlefoc-state type.
type lmf-singlefoc-cert lmf-singlefoc-state -> lmf-tree -> cert.
type lmf-singlefoc-node index -> index -> lmf-node.
```

In `Checkers`, proof objects are elements of type `cert`. The different "state" elements are used internally by the FPC specifications and are initialized to default values.

The main component of the proof evidence are the nodes. For *LMF*, we require information about two indices. The first is the index of the principal formula in the inference and the second is an optional index. This index is mandatory in the following two cases:

- If the principal formula is a $\Diamond$, then the second index must be the index of the associated box formula.
- If the inference is an `init` rule, then the second index is the index of the complementary literal.

In addition to the type declaration, the FPC definition must supply the logic program defining the clerk predicates and the expert predicates. Writing no specification for a given predicate defines that predicate to hold for no list of arguments.

According to this specification, which can be found in [19], each decide step is completely determined by the proof evidence.

4.1.2 The *LMF*^{m} system layer

This layer allows us to simulate a multi-focusing step in the kernel and corresponds to the multi-focused version of *LMF* defined in section 3.2. Our system will simulate multi-focusing using a non multi-focusing kernel by relating each inference with a number. This number will force all inferences labeled the same to occur sequentially. This does not simulate multi-focusing in the general case, since in our implementation processing one of the formulas does modify the state in which a second formula is processed. However, the fact that we only multifocus on $\Diamond$-formulas on a given label and the fact that we restrict to the logic K ensure that the simple mechanism defined above is enough for encoding multifocusing in our case.

In addition to the proof format from the previous layer, we require every node to contain a multi-focus value.

```
% defined in lmf-multifoc.sig
type lmf-multifoc-node int -> lmf-node -> lmf-node.
```

4.1.3 The *LMF** system layer

The most expressive layer is *LMF** which directly corresponds to the LMF_* system defined in section 3.2. This layer extends the previous one with information about worlds which are

```
% defined in lmf-star.sig
type lmf-star_to_lmf-multifoc cert -> lmf-star-state -> list A -> A -> cert -> o.
type lmf-multifoc_to_lmf-star cert -> lmf-star-state -> list A -> A -> cert -> o.

% defined in lmf-star.mod
lmf-star_to_lmf-multifoc
  (lmf-star-cert S (lmf-multifoc-cert (lmf-singlefoc-cert S1
    (lmf-tree (lmf-star-node H F N) C))))
  S H F
  (lmf-multifoc-cert (lmf-singlefoc-cert S1 (lmf-tree N C))).
lmf-star_to_lmf-multifoc
  (lmf-star-cert S (lmf-multifoc-cert (lmf-singlefoc-cert S1 (lmf-tree N C))))
  S H F
  (lmf-multifoc-cert (lmf-singlefoc-cert S1 (lmf-tree N C))).
lmf-multifoc_to_lmf-star
  (lmf-multifoc-cert (lmf-singlefoc-cert S1 (lmf-tree (lmf-multifoc-node M N) C)))
  S H F
  (lmf-star-cert S (lmf-multifoc-cert (lmf-singlefoc-cert S1
    (lmf-tree (lmf-star-node H F (lmf-multifoc-node M N)) C)))).
lmf-multifoc_to_lmf-star
  (lmf-multifoc-cert (lmf-singlefoc-cert S1 (lmf-tree N C)))
  S _ _
  (lmf-star-cert S (lmf-multifoc-cert (lmf-singlefoc-cert S1 (lmf-tree N C)))).
```

Figure 8: Proof evidence transformation between two layers

currently active (the present) and the possible futures. The present is intuitively used in order to restrict the application of the decide rule only to formulas labeled by nodes contained in the present. The future can be used to restrict the application of the $\Diamond$ rule. Please refer to section 3.2 for more information.

The supported proof format for this layer is denoted by the following types:

```
% defined in lmf-star.sig
kind lmf-star-state type.
type lmf-star-state list A -> A -> list (pair index A) -> lmf-star-state.
type lmf-star-node list A -> A -> lmf-node -> lmf-node.
type lmf-star-cert lmf-star-state -> cert -> cert.
```

The state is now extended to contain information about the current present and future, as well as information about the actual label assigned to each index. The nodes of a proof evidence, as defined in section 3.2, contain, in addition to the information required in the previous layer, also information about the new present and future.

In the structure of the evidence accepted in this layer, one can see the abstraction and polymorphism mechanism applied in `Checkers`. An `lmf-node` is an abstract type which corresponds to all concrete implementations of the nodes. In order to fake polymorphism, we have implemented a set of transformations between the different layers, as can be seen in figure 8.

Given a proof evidence in the format supported by one layer, the FPC specification will recursively apply the specification defined for the lower layer, using transformations similar to the ones in Figure 8. The expressiveness of the upper layer will be used in order to prune nodes in the search space, as well as for sometimes changing the information passed to the lower level.

Figure 9 gives the FPC specifications for the *LMF** layer. The auxiliary relations used are the following:

- `obtaine_all_star_node_vals` is used to extract the values in the state and root node
- `obtain_value_in_map` returns the label associated to a given index
- `member` checks for list membership
- `change_state` updates the state in a certificate

The FPC specifications in Figure 9 corresponds to the definition of LMF_* in Section 3.2. For example, the `decide_ke` expert calls the lower layer only in case the world associated with the root node is allowed by the present in the inference rule.

4.2 Certification of different proof formats

Given the different layers in the proof system defined in the previous section, we can easily write FPC specifications for different popular proof formats.

The process is always the same. The FPC specifications for the proof format translates the root node of the proof evidence into the format of a node in one of the layers of the framework and make a recursive call. In case the inference of the input calculus corresponds to exactly one inference in the framework calculus, the result of the recursive call is an inference tree whose new root node is again a node in the input calculus. In other cases, the translations between the layers will make sure to use the right type of nodes in order to imitate several steps within the framework.

In the next sections we describe in more detail how the framework is used in order to support specific proof formats.

4.2.1 Labeled sequents

The treatment of labeled systems [24] was already implemented in the previous version of `Checkers`, which is described in [19]. In order to get emulation of *LS*, we require a very simple use of the framework LMF_*, where at each node the present of a sequent corresponds to the set of all the labels occurring in the proof, no use of multifocusing is required and the future of a node is set, in the case of $\Diamond$-formulas, to the index of the corresponding $\Box$-formula. For simplicity, since this is enough in the case of K, in our implementation we rely on the lower layer *LMF*. Please refer to [19].

4.2.2 Prefixed tableaux

The popular PT proof format [9], which is used by various automated theorem provers, is, in the case of K, very close to that of *LS* (we can roughly say that PT, being a refutation method, is the dual of *LS*). Therefore support for it can be obtained in a very similar way. Its implementation, which has been described in [19], also relies on *LMF* and mainly consists in inverting, with respect to *LS* the role of boxes and diamonds in the FPC and in letting tableau closure rules behave as sequent initial rules.

4.2.3 Ordinary Sequents

As described in Section 2.3.5, ordinary sequent systems differ in several ways from the previous systems. First, they do not have labels and second, they treat both $\Box$ and $\Diamond$-formulas inside a single inference rule. For these reasons, the case of ordinary sequents illustrates the use of the features of the framework LMF_* in a more significant way already for the logic K.

In particular, the modal rule, which applies to all $\Diamond$-formulas at once, can be emulated in our system by using multi-focusing. In addition, the relationship between the modal operators can be used in order to restrict the futures allowed: given a modal rule, all the $\Diamond$-formulas there occurring are assigned the same future, which corresponds to the index of the only $\Box$-formula.

Next, we specify the expected format of ordinary sequent proof evidences.

```
% defined in lmf-star.mod
decide_ke Cert L Cert' :-
  obtain_all_star_node_vals Cert H F Map NH NF M I OI,
  obtain_value_in_map Map I V,
  member V H,
  lmf-star_to_lmf-multifoc Cert S NH NF Cert-s,
  decide_ke Cert-s L Cert-s',
  lmf-multifoc_to_lmf-star Cert-s' (lmf-star-state NH NF Map) NH NF Cert'.
store_kc Cert L B Cert' :-
  lmf-star_to_lmf-multifoc Cert S H F Cert-s,
  store_kc Cert-s L B Cert-s',
  lmf-multifoc_to_lmf-star Cert-s' S H F Cert'.
release_ke Cert Cert.
initial_ke Cert O :-
  lmf-star_to_lmf-multifoc Cert _ _ _ Cert-s,
  initial_ke Cert-s O.
orNeg_kc Cert Form Cert' :-
  lmf-star_to_lmf-multifoc Cert S H F Cert-s,
  orNeg_kc Cert-s Form Cert-s',
  lmf-multifoc_to_lmf-star Cert-s' S H F Cert',
  obtain_all_star_node_vals Cert _ _ Map _ _ _ I _,
  obtain_all_star_node_vals Cert' _ _ Map' _ _ _ I _,
  obtain_value_in_map Map I V.
orNeg_kc Cert Form Cert-r :-
  lmf-star_to_lmf-multifoc Cert S H F Cert-s,
  orNeg_kc Cert-s Form Cert-s',
  lmf-multifoc_to_lmf-star Cert-s' S H F Cert',
  obtain_all_star_node_vals Cert _ _ Map _ _ _ I _,
  obtain_value_in_map Map I V,
  add_value_to_map_in_state S V (lind I) S',
  add_value_to_map_in_state S' V (rind I) S'',
  change_state Cert' S'' Cert-r.
andNeg_kc Cert Form Cert1-r Cert2-r :-
  lmf-star_to_lmf-multifoc Cert S H F Cert-s,
  andNeg_kc Cert-s Form Cert-s1 Cert-s2,
  lmf-multifoc_to_lmf-star Cert-s1 S H F Cert1,
  lmf-multifoc_to_lmf-star Cert-s2 S H F Cert2,
  obtain_all_star_node_vals Cert H F Map NH NF M I OI,
  obtain_value_in_map Map I V,
  add_value_to_map_in_state S1 V (lind I) S1b,
  add_value_to_map_in_state S2 V (rind I) S2b,
  change_state Cert1 S1b Cert1-r,
  change_state Cert2 S2b Cert2-r.
andPos_k Cert Form Str Cert1 Cert2 :-
  lmf-star_to_lmf-multifoc Cert S H F Cert-s,
  andPos_k Cert-s Form Str Cert-s1 Cert-s2,
  lmf-multifoc_to_lmf-star Cert-s1 S H F Cert1,
  lmf-multifoc_to_lmf-star Cert-s2 S H F Cert2.
all_kc (lmf-star-cert State Cert) Cert' :-
  lmf-star_to_lmf-multifoc (lmf-star-cert State Cert) S _ _ Cert-s,
  all_kc Cert-s Cert-s',
  obtain_all_star_node_vals (lmf-star-cert State Cert) H F Map NH NF M I OI,
  add_value_to_map_in_state S I I S',
  lmf-multifoc_to_lmf-star_all Cert-s' S' Cert'.
some_ke Cert X Cert'-r :-
  lmf-star_to_lmf-multifoc Cert S H F Cert-s,
  some_ke Cert-s X Cert-s',
  lmf-multifoc_to_lmf-star Cert-s' S' H F Cert',
  obtain_all_star_node_vals Cert H F Map NH F M I OI,
  add_value_to_map_in_state S (OI) (diaind I OI) S1,
  change_state Cert' S1 Cert'-r.
```

Figure 9: FPC specifications for the LMF^* layer

```
% defined in ordinary-sequents.sig
type ordinary-sequent-node index -> list index -> lmf-node.
type ordinary-sequent-cert ordinary-sequent-state -> lmf-tree -> cert.
```

An ordinary sequent node contains its index as well as a list of indices. This list is empty for all inference rules except for the modal rule, where it specifies the indices of all the ◊-formulas that are affected, as well as for the initial rule, in which case the list contains a single index denoting the complementary literal.

As discussed above, the state is not an integral part of the proof evidence and is therefore omitted.

The FPC specification for ordinary sequents is easily implemented on top of the *LMF** layer. The only two non-trivial steps relate to the modal rule. On reaching such an inference rule, we translate the evidence, in a similar way to the one shown in Figure 8, to the *LMF** layer. Before we make the recursive call, we modify the tree in the evidence by adding a multifocusing node (emulated by a sequence of nodes decorated by the same multifocusing value) that contains all the ◊-formulas. Another non-trivial step occurs when we reach the last ◊-formula in such a sequence. Since there is no inference rule for these formulas in the ordinary sequent calculus, we need to translate back from *LMF** to ordinary sequent at the right point. This is taken care of by the `decide` expert. In other words, we only return the control back to the ordinary sequent layer when all the ◊-formulas have been processed.

The FPC specifications are given in Figure 10. The auxiliary relations used are:

- `ordinary-sequent-to-lmf-star`, which translates between the general framework and the ordinary sequent layer;
- `generate_diamonds`, which generates the inferences corresponding to the ◊-formulas, to be added to the tree.

4.2.4 Nested Sequents

A more challenging example of using our framework is supporting nested sequent proof evidence. Here we will also demonstrate how simpler layers can be used in order to support proof formats.

When considering the nested sequent proof system for K, we notice that ◊-formulas are associated to specific □-formulas. This property does not hold in general.

This association is similar to the one in *LMF* and allows us to use this simpler layer for the support of the proof evidence.

Our format for nested sequents is given in figure 11. Note that we now index formulas using two separate indices: The first one is just the location of the sub-formula while the second is the branch of the nested sequent.

The formal definition of indices of nested sequents is given next.

Definition 2.1 (Indexing Nested Sequents)**.** *Indices of nested sequents are defined recursively by:*

- `zb` *is an index (of the top level nested sequent).*
- *if* `ind` *is an index of a nested sequent and we have in it a nested sequent at the* i^{th} *position, then* `(chld i ind)` *is an index denoting this nested sequent.*

Figure 12 gives an example of a nested sequent derivation and the indices of sub-formulas.

In order to certify nested sequent proofs in our framework, we will use, as mentioned above, the *LMF* layer. This layer requires a correspondence between ◊-formulas and □-formulas.

```
% defined in ordinary-sequents.mod
decide_ke
  (lmf-star-cert (lmf-star-state H F Map) (lmf-multifoc-cert (lmf-singlefoc-cert
    (lmf-singlefoc-state IL Eig) (lmf-tree (ordinary-sequent-node I OI) C))))
  L
  Cert' :-
  decide_ke
    (ordinary-sequent-cert
      (ordinary-sequent-state H F Map 0 IL Eig) (lmf-tree
        (ordinary-sequent-node I OI) C))
      L
      Cert'.
decide_ke Cert L Cert' :-
  ordinary-sequent-to-lmf-star Cert IO Cert-s,
  decide_ke Cert-s L Cert-s',
  lmf-star-to-ordinary-sequent Cert-s' IO Cert'.
store_kc Cert L B Cert' :-
ordinary-sequent-to-lmf-star Cert OI Cert-s,
store_kc Cert-s L B Cert-s',
lmf-star-to-ordinary-sequent Cert-s' OI Cert'.
release_ke Cert Cert.
initial_ke Cert O :-
  ordinary-sequent-to-lmf-star-with-op-index Cert Cert-s,
  initial_ke Cert-s O.
orNeg_kc Cert Form Cert-r :-
 ordinary-sequent-to-lmf-star Cert IO Cert-s,
 orNeg_kc Cert-s Form Cert-s',
 lmf-star-to-ordinary-sequent Cert-s' IO Cert-r.
andNeg_kc Cert Form Cert1 Cert2 :-
  ordinary-sequent-to-lmf-star Cert IO Cert-s,
  andNeg_kc Cert-s Form Cert1' Cert2',
  lmf-star-to-ordinary-sequent Cert1' IO Cert1,
  lmf-star-to-ordinary-sequent Cert2' IO Cert2.
all_kc
  ordinary-sequent-cert
    (ordinary-sequent-state H F Map _ IL Eig)
      (lmf-tree (ordinary-sequent-node I OI) C)
   Cert-r :-
  generate_diamonds I OI C T H F 0,
  all_kc
    (lmf-star-cert (lmf-star-state H F Map)
      (lmf-multifoc-cert
        (lmf-singlefoc-cert (lmf-singlefoc-state IL Eig)
          (lmf-tree (lmf-star-node H F (lmf-multifoc-node 0
            (lmf-singlefoc-node I none))) T))))
    Cert-r.
```

Figure 10: FPC specifications for ordinary sequents

```
% defined in nested-sequents.sig
type ns index -> index -> index.
type chld int -> index -> index.
type zb index.
type nested-sequent-node index -> index -> lmf-node.
type nested-sequent-cert nested-sequent-state -> lmf-tree -> cert.
```

Figure 11: Type definitions on nested sequents

$$\cfrac{\cfrac{(q)^{\texttt{(ns (lind root) zb)}}, ([p])^{\texttt{(ns (rind root) (chld 1 zb))}}}{(q)^{\texttt{(ns (lind root) zb)}}, (\Box p)^{\texttt{(ns (rind root) zb)}}}}{(q \vee \Box p)^{\texttt{(ns root zb)}}}$$

Figure 12: An example of a nested sequent derivation and the corresponding indices

Since there is such a correspondence in *NS*, our implementation of the FPC specifications for this system tries to exploit it. This will be done by mapping the indices of one system to the indices of the other.

The mapping of the indices as well as some other similar data structures will be stored in the state.

Figure 13 shows our implementation where the auxiliary functions are:

- `convert-index` which converts indices from nested sequents to *LKF* using a map.
- `add_to_map` adds new indices to the map.
- `get_incremented_child` increases the counter associated with a certain index.

As can be seen, supporting nested sequent proof evidence for K is straightforward and does not require any knowledge of *LKF*. The only thing required is to be able to translate between the indices.

One can also observe that we do have one non-trivial manipulation in the implementation. The `all_kc` definition does not depend on the one in `lmf-singlefoc` but instead re-implement it. The reason for that is the inability of λProlog to unify objects of functional type. We hope to get around that in future versions.

4.3 Examples

In this section, we apply the specifications from the previous section to several examples. The examples consist of a hand-generated proof evidence in few formats of the validity of the K axiom: $\Diamond(P \wedge \neg Q) \vee \Diamond\neg P \vee \Box Q$.

The examples in this section and others can be found in the testing section of the `Checkers` proof certifier. `Checkers` can be obtained online [3] and can be executed by running in a bash terminal: [4].

```
$ ./prover.sh arg
```

where the argument is the name of the λProlog module denoting the proof evidence one wishes to check.

In Figure 14, one can see the proof evidence corresponding to the ordinary sequent proof of Figure 15.

Another example is given in figure 16. The proof which generates this nested sequent example can be seen in figure 17.

Please refer to the `src/test/modal` folder for more examples.

[3]The exact version can be found on the "framework" branch in the git repository https://github.com/proofcert/checkers.

[4]`Checkers` depends on the λProlog interpreter Teyjus (http://teyjus.cs.umn.edu/)

```
% defined in nested-sequents.mod
decide_ke Cert I' Cert' :-
  Cert = (nested-sequent-cert (nested-sequent-state Counter Map V M)
    (lmf-tree (nested-sequent-node I O) D)),
  convert-index Map I I-s,
  nested-sequent-to-lmf-singlefoc Cert I-s Cert-s,
  decide_ke Cert-s I' Cert-s',
  lmf-singlefoc-to-nested-sequent Cert-s' Counter Map I Cert'.
store_kc Cert Form H Cert' :-
  Cert = (nested-sequent-cert (nested-sequent-state Counter Map V M)
    (lmf-tree (nested-sequent-node I O) D)),
  convert-index Map I I-s,
  nested-sequent-to-lmf-singlefoc Cert I-s Cert-s,
  store_kc Cert-s Form H Cert-s',
  lmf-singlefoc-to-nested-sequent Cert-s' Counter Map I Cert'.
release_ke C C.
initial_ke (nested-sequent-cert (nested-sequent-state Counter Map V M)
  (lmf-tree (nested-sequent-node I O) D)) O' :-
  convert-index Map O O'.
orNeg_kc Cert Val Cert' :-
  Cert = (nested-sequent-cert (nested-sequent-state Counter Map V M)
    (lmf-tree (nested-sequent-node I O) D)),
  convert-index Map I I-s,
  I = (ns Ind Ch),
  nested-sequent-to-lmf-singlefoc Cert I-s Cert-s,
  orNeg_kc Cert-s Val Cert-s',
  Cert-s' = (lmf-singlefoc-cert (lmf-singlefoc-state [I1,I2] _) _),
  add_to_map Map (ns (lind Ind) Ch) I1 Map1,
  add_to_map Map1 (ns (rind Ind) Ch) I2 Map2,
  lmf-singlefoc-to-nested-sequent Cert-s' Counter Map2 _ Cert'.
andNeg_kc Cert _ Cert1 Cert2 :-
  Cert = (nested-sequent-cert (nested-sequent-state Counter Map V M)
    (lmf-tree (nested-sequent-node I O) D)),
  convert-index Map I I-s,
  I = (ns Ind Ch),
  nested-sequent-to-lmf-singlefoc Cert I-s Cert-s,
  andNeg_kc Cert-s _ Cert-s1 Cert-s2,
  Cert-s1 = (lmf-singlefoc-cert (lmf-singlefoc-state [I1] _) _),
  add_to_map Map (ns (lind Ind) Ch) I1 Map1,
  lmf-singlefoc-to-nested-sequent Cert-s1 Counter Map1 _ Cert1,
  Cert-s2 = (lmf-singlefoc-cert (lmf-singlefoc-state [I2] _) _),
  add_to_map Map (ns (rind Ind) Ch) I2 Map2,
  lmf-singlefoc-to-nested-sequent Cert-s2 Counter Map2 _ Cert2.
andPos_k Cert _ Stra Cert1 Cert2 :-
  Cert = (nested-sequent-cert (nested-sequent-state Counter Map V M)
    (lmf-tree (nested-sequent-node I O) D)),
  convert-index Map I I-s,
  I = (ns Ind Ch),
  nested-sequent-to-lmf-singlefoc Cert I-s Cert-s,
  andPos_k Cert-s _ Stra Cert-s1 Cert-s2,
  Cert-s1 = (lmf-singlefoc-cert (lmf-singlefoc-state [I1] _) _),
  add_to_map Map (ns (lind Ind) Ch) I1 Map1,
  lmf-singlefoc-to-nested-sequent Cert-s1 Counter Map1 _ Cert1,
  Cert-s2 = (lmf-singlefoc-cert (lmf-singlefoc-state [I2] _) _),
  add_to_map Map (ns (rind Ind) Ch) I2 Map2,
  lmf-singlefoc-to-nested-sequent Cert-s2 Counter Map2 _ Cert2.
all_kc Cert
  (Eigen\ nested-sequent-cert (nested-sequent-state NewCounter
    Map' [lind I-s] [pr I-s Eigen|M]) D) :-
  Cert = (nested-sequent-cert (nested-sequent-state Counter Map [] M)
    (lmf-tree (nested-sequent-node I O) [D])),
  convert-index Map I I-s,
  I = (ns Ind Ch),
  get_incremented_child Counter Ch NewCh NewCounter,
  nested-sequent-to-lmf-singlefoc Cert I-s Cert-s,
  add_to_map Map (ns I NewCh) (lind I-s) Map'.
some_ke Cert X Cert' :-
  Cert = (nested-sequent-cert (nested-sequent-state Counter Map V M)
    (lmf-tree (nested-sequent-node I O) D)),
  convert-index Map I I-s,
  nested-sequent-to-lmf-singlefoc Cert I-s Cert-s,
  some_ke Cert-s X Cert-s',
  Cert-s' = (lmf-singlefoc-cert (lmf-singlefoc-state [I'] _) _),
  add_to_map Map (ns I O) I' Map',
  lmf-singlefoc-to-nested-sequent Cert-s' Counter Map' _ Cert'.
```

Figure 13: FPC specifications for nested sequents

```
module ex-os1.                                                    % module declaration
accumulate ordinary-sequents.                                     % fpc specification
accumulate lkf-kernel.                                            % kernel in use
accumulate modal-encoding.                                        % modal-translation
modalProblem "The␣K␣Axiom"                                        % problem description
(((dia (-- p1)) !! (box (++ q1))) !! (dia ((++ p1) && (-- q1)))) % modal theorem
(ordinary-sequent-cert                                            % proof evidence
  (ordinary-sequent-state [root] none [pr root root] 0 [] [])
  (lmf-tree (ordinary-sequent-node root [none]) [
    lmf-tree (ordinary-sequent-node (rind root) [none]) [
      lmf-tree (ordinary-sequent-node (rind (rind root)) [(lind (rind root)),
          (lind root)]) [
        lmf-tree (ordinary-sequent-node (diaind (lind root) (rind (rind root))) [none]) [
          lmf-tree (ordinary-sequent-node (diaind (lind (rind root)) (rind (rind root)))
              [none]) [
            lmf-tree (ordinary-sequent-node (lind (diaind (lind root) (rind (rind root))))
                [diaind (lind(rind root)) (rind (rind root))]) []],
          lmf-tree (ordinary-sequent-node (rind (diaind (lind root) (rind (rind root))))
              [none]) [
            lmf-tree (ordinary-sequent-node (lind (rind (rind root))) [(rind (diaind
                (lind root) (rind(rind root))))]) []]]]]]]).
```

Figure 14: src/test/modal/ex-os1.mod

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash Q, \neg Q \qquad \vdash P, \neg P}{\vdash P \wedge \neg Q, \neg P, Q}}{\vdash \Diamond(P \wedge \neg Q), \Diamond\neg P, \Box Q}}{\vdash \Diamond(P \wedge \neg Q) \vee \Diamond\neg P, \Box Q}}{\vdash \Diamond(P \wedge \neg Q) \vee \Diamond\neg P \vee \Box Q}$$

Figure 15: Ordinary sequent proof of axiom K

5 Conclusion

We have presented here an implementation of a framework for certifying proofs produced in several modal proof formalisms. The framework has been developed by following the general principles of the project ProofCert and as a module of the concrete implementation provided by Checkers. Such an implementation uses an augmented version of the focused classical sequent system *LKF* as a kernel. The augmentation is obtained by enriching the calculus with predicates able to reconstruct an original proof by following the information given in the evidence. In a sense, we can see our framework as a bridge between modal proof systems and *LKF*. Such a bridge is obtained by restricting the power of these further predicates to the minimal needed in the context of modal logics.

There are several ways in which this work can be extended. The design of the parametric devices of the framework has been driven by the ambition of being as comprehensive as possible in terms of formalisms captured. The modularity and parameterizability of the whole approach should make it possible, in fact, to consider other related approaches to modal proof theory, like hypersequent calculi [2], e.g., by using a *present* parameter that is a multiset, representing external structural rules as operations on such a present, and viewing modal communication rules as a combination of relational and modal rules. The focused nature of the approach should also allow for certifying proofs coming from focused proof systems for modal logics, like the ones in [17, 7], possibly by using a different polarization of formulas.

```
module ex-nseq1.
accumulate nested-sequents.
accumulate lkf-kernel.
accumulate modal-encoding.
modalProblem "Problem:␣Axiom␣K␣for␣nested-sequents"
((dia((++ p1) && (-- q1))) !! ((dia(-- p1)) !! (box (++ q1))))
(nested-sequent-cert
 (nested-sequent-state [pr zb 0] [pr (ns root zb) root] [] [])
  (lmf-tree (nested-sequent-node (ns root zb)  none) [
   (lmf-tree (nested-sequent-node (ns (rind root) zb) none) [
    (lmf-tree (nested-sequent-node (ns (rind (rind root)) zb) none) [
     (lmf-tree (nested-sequent-node (ns (lind (rind root)) zb) (chld 1 zb)) [
      (lmf-tree (nested-sequent-node (ns (lind root) zb) (chld 1 zb)) [
       (lmf-tree (nested-sequent-node (ns (lind root) (chld 1 zb)) none) [
        (lmf-tree (nested-sequent-node (ns (rind (lind root)) (chld 1 zb)) none) [
         (lmf-tree (nested-sequent-node (ns (rind (rind root)) (chld 1 zb))
            (ns (rind (lind root)) (chld 1 zb))) [])]) ,
        (lmf-tree (nested-sequent-node (ns (lind (rind root)) (chld 1 zb)) none ) [
         (lmf-tree (nested-sequent-node (ns (lind (lind root)) (chld 1 zb))
            (ns (lind (rind root)) (chld 1 zb))) [])])])])])])])])).
```

Figure 16: src/test/modal/ex1-nseq1.mod

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Diamond(P \wedge \neg Q), \Diamond\neg P, [\neg Q, \neg P, Q] \qquad \Diamond(P \wedge \neg Q), \Diamond\neg P, [P, \neg P, Q]}{\Diamond(P \wedge \neg Q), \Diamond\neg P, [P \wedge \neg Q, \neg P, Q]}}{\Diamond(P \wedge \neg Q), \Diamond\neg P, [\neg P, Q]}}{\Diamond(P \wedge \neg Q), \Diamond\neg P, [Q]}}{\Diamond(P \wedge \neg Q), \Diamond\neg P, \Box Q}}{\Diamond(P \wedge \neg Q), \Diamond\neg P \vee \Box Q}}{\Diamond(P \wedge \neg Q) \vee (\Diamond\neg P \vee \Box Q)}$$

Figure 17: Nested sequent proof of axiom K

Orthogonally, we also aim at extending the approach to variants of the logic K. This can be done, at least for the logics characterized by the so-called geometric frames, according to the recipes provided in [20].

Finally, we remark that while this work was inspired by certification consisting in a strict emulation of original proofs, it is sometimes the case that only partial information about the proof to be checked is provided. We plan to complement the current implementation with a “relaxed” version of the FPCs, such that it can also deal with incomplete proof evidences, similarly to what has been done in [19] in order to check, e.g., free-variable tableau [3] proofs.

Acknowledgment. This work was funded by the ERC Advanced Grant ProofCert.

References

[1] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *J. Log. Comput.*, 2(3):297–347, 1992.

[2] Arnon Avron. The method of hypersequents in the proof theory of propositional non-classical logics. In *Logic: From Foundations to Applications, European Logic Colloquium*, pages 1–32. Oxford University Press, 1994.

[3] Bernhard Beckert and Rajeev Goré. Free-variable tableaux for propositional modal logics. *Studia Logica*, 69(1):59–96, 2001.

[4] Patrick Blackburn and Johan Van Benthem. Modal logic: a Semantic Perspective. In Frank Wolter Patrick Blackburn, Johan van Benthem, editor, *Handbook of Modal Logic*, pages 1–82. Elsevier, 2007.

[5] Kai Brünnler. Deep sequent systems for modal logic. *Archive for Mathematical Logic*, 48(6):551–577, 2009.

[6] Kaustuv Chaudhuri, Stefan Hetzl, and Dale Miller. A multi-focused proof system isomorphic to expansion proofs. *J. Log. Comput.*, 26(2):577–603, 2016.

[7] Kaustuv Chaudhuri, Sonia Marin, and Lutz Straßburger. Focused and synthetic nested sequents. In Bart Jacobs and Christof Löding, editors, *FoSSaCS*, pages 390–407, 2016.

[8] Zakaria Chihani, Tomer Libal, and Giselle Reis. The proof certifier checkers. In Hans de Nivelle, editor, *Automated Reasoning with Analytic Tableaux and Related Methods - 24th International Conference, TABLEAUX 2015, Wrocław, Poland, September 21-24, 2015. Proceedings*, volume 9323 of *Lecture Notes in Computer Science*, pages 201–210. Springer, 2015.

[9] Melvin Fitting. Tableau methods of proof for modal logics. *Notre Dame Journal of Formal Logic*, 13(2):237–247, 1972.

[10] Melvin Fitting. Modal proof theory. In Patrick Blackburn, Johan van Benthem, and Frank Wolter, editors, *Handbook of Modal Logic*, pages 85–138. Elsevier, 2007.

[11] Melvin Fitting. Prefixed tableaus and nested sequents. *Ann. Pure Appl. Logic*, 163(3):291–313, 2012.

[12] Dov M. Gabbay. *Labelled Deductive Systems*. Clarendon Press, 1996.

[13] Rajeev Goré and Revantha Ramanayake. Labelled tree sequents, tree hypersequents and nested (deep) sequents. In *Advances in Modal Logic 9, papers from the ninth conference on "Advances in Modal Logic," held in Copenhagen, Denmark, 22-25 August 2012*, pages 279–299, 2012.

[14] Andrzej Indrzejczak. *Natural Deduction, Hybrid Systems and Modal Logics*. Springer, 2010.

[15] Ryo Kashima. Cut-free sequent calculi for some tense logics. *Studia Logica*, 53(1):119–136, 1994.

[16] Björn Lellmann. Linear nested sequents, 2-sequents and hypersequents. In Hans de Nivelle, editor, *Automated Reasoning with Analytic Tableaux and Related Methods - 24th International Conference, TABLEAUX 2015, Wrocław, Poland, September 21-24, 2015. Proceedings*, volume 9323 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 2015.

[17] Björn Lellmann and Elaine Pimentel. Proof search in nested sequent calculi. In Martin Davis, Ansgar Fehnker, Annabelle McIver, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings*, volume 9450 of *Lecture Notes in Computer Science*, pages 558–574. Springer, 2015.

[18] Chuck Liang and Dale Miller. Focusing and polarization in linear, intuitionistic, and classical logics. *Theor. Comput. Sci.*, 410(46):4747–4768, 2009.

[19] Tomer Libal and Marco Volpe. Certification of prefixed tableau proofs for modal logic. In *Proceedings of the Seventh International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2016, Catania, Italy, 14-16 September 2016.*, pages 257–271, 2016.

[20] Sonia Marin, Dale Miller, and Marco Volpe. A focused framework for emulating modal proof systems. In *Proceedings of the 11th International Conference on Advances in Modal Logic, Budapest, 30 August - 2 September 2016*, 2016. To appear.

[21] Dale Miller. Proofcert: Broad spectrum proof certificates. An ERC Advanced Grant funded for the five years 2012-2016, February 2011.

[22] Dale Miller and Gopalan Nadathur. *Programming with Higher-Order Logic*. Cambridge University Press, 2012.

[23] Dale Miller and Marco Volpe. Focused labeled proof systems for modal logic. In Martin Davis,

Ansgar Fehnker, Annabelle McIver, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings*, volume 9450 of *Lecture Notes in Computer Science*, pages 266–280. Springer, 2015.

[24] Sara Negri. Proof analysis in modal logic. *J. Philosophical Logic*, 34(5-6):507–544, 2005.

[25] Francesca Poggiolesi. *Gentzen Calculi for Modal Propositional Logic*. Springer, 2011.

[26] Charles Stewart and Phiniki Stouppa. A systematic proof theory for several modal logics. In *5th Conference on "Advances in Modal logic," Manchester (UK), September 2004*, pages 309–333, 2004.