Certification of Prefixed Tableau Proofs for Modal Logic

Tomer Libal Marco Volpe
Inria and LIX, École Polytechnique
France
tomer.libal@inria.fr marco.volpe@inria.fr

Different theorem provers tend to produce proof objects in different formats and this is especially the case for modal logics, where several deductive formalisms (and provers based on them) have been presented. This work falls within the general project of establishing a common specification language in order to certify proofs given in a wide range of deductive formalisms. In particular, by using a translation from the modal language into a first-order polarized language and a checker whose small kernel is based on a classical focused sequent calculus, we are able to certify modal proofs given in labeled sequent calculi, prefixed tableaux and free-variable prefixed tableaux. We describe the general method for the logic K, present its implementation in a Prolog-like language, provide some examples and discuss how to extend the approach to other normal modal logics.

1 Introduction

Modal logics are very popular and feature in many areas of computer science, including formal verification, knowledge representation, the field of logics of programs, computational linguistics and agent-based systems. Two common approaches for the automatic proving of modal theorems are the tableau method [9] and the resolution principle [19]. Theorem provers based on such approaches normally contain non-trivial optimizations and cores which might compromise the amount of trust we can place in them. Nevertheless, only few of these provers do actually return an evidence supporting their results and even these evidences might not be checkable by a computer.

ProofCert [15] is a project whose main goal is the certification of a wide range of proof evidences. 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. The certification is carried out by using dedicated FPC specifications which guide the construction of proofs in the target kernels. A particularly trusted and low-level kernel is the focused classical sequent calculus *LKF* [13]. In [17], a translation from the language of the labeled sequent system *G3K* [18] for propositional modal logic into the language of *LKF* was described. *G3K* is of interest when trying to certify proofs of modal theorem provers due to its close relationship with the refutational technique of prefixed tableaux, on which many modal theorem provers are based.

In this paper, we propose two distinct FPC specifications, both relying on such a translation. The first one requires a detailed proof description from the prover and allows for a step-by-step checking, while the second one only needs some core information about the original proof and operates by performing some proof reconstruction. These specifications enable the automated checking of proof evidences coming

D. Cantone and G. Delzanno (Eds.): Seventh Symposium on
Games, Automata, Logics and Formal Verification (GandALF'16)
EPTCS 226, 2016, pp. 257–271, doi:10.4204/EPTCS.226.18

from different deductive formalisms for the modal logic K. In particular, we will show how to apply them to the certification of proofs given in *G3K*, in Fitting-style prefixed tableaux and in a free-variable variant of prefixed tableau systems. While the first calculus is designed for positive proofs, the other two are based on a refutational method. Still, the most dramatic change in the form of the derivations is presented in the third method, as the free-variable optimization introduces the notion of meta-variables and can significantly change the structure of the proofs generated. Proof evidences arising from these formalisms, when paired with the corresponding specification, can be automatically certified by Checkers over the *LKF* kernel. We show, by means of examples, that using the ProofCert flexible notion of a proof evidence and Checkers modular design, we are able to support proof checking for these three different formalisms, by making use of the same translation.

To the best of our knowledge, the work presented here is the first attempt to independently certify the proofs generated by propositional modal theorem provers. The approach closest to ours is probably Dedukti's [6] independent certification for the classical first-order tableau prover Zenon modulo [7].

In the next section, we present some background on ProofCert, modal logic and theorem proving. In Section 3, we describe the different FPC specifications. Such specifications are then used in order to enhance the capabilities of Checkers, as we demonstrate on some examples. In Section 4, 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 [16] 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 3.
- **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 3.

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 [13]. 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 subformula 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 $\partial^+(A) = \mathtt{true} \wedge^+ A$ and $\partial^-(A) = \mathtt{false} \vee^- A$. 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 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 3 is mainly devoted to the definitions of these programs for the different proof formats of the modal theorem provers.

2.3 Prefixed tableaux for modal 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

INVERTIBLE RULES

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

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

FOCUSED RULES

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

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

IDENTITY RULES

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

STRUCTURAL RULES

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

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

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.

set $\mathscr{P}$ 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 $P \in \mathscr{P}$. 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 $\mathscr{F} = (W, R)$ where W is a non empty set of *worlds* and R is a binary relation on W. A *Kripke model* is a triple $\mathscr{M} = (W, R, V)$ where (W, R) is a Kripke frame and $V : W \to 2^{\mathscr{P}}$ is a function that assigns to each world in W a (possibly empty) set of propositional symbols.

Truth of a modal formula at a point w in a Kripke structure $\mathscr{M} = (W, R, V)$ is the smallest relation $\models$ satisfying:

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

CLASSICAL RULES

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

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

MODAL RULES

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

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

Figure 2: *G3K*: a labeled sequent system for the modal logic *K*

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

2.3.2 The standard translation from modal logic into classical logic

The following *standard translation* (see, e.g., [5]) provides a bridge between propositional modal logic and first-order classical logic:

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

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* [5] and consists of a binary predicate symbol R and a unary predicate symbol P for each $P\in\mathscr{P}$. 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 $\mathscr{M}$ and any world w, we have that $\mathscr{M},w\models A$ if and only if $\mathscr{M}\models ST_x(A)[x\leftarrow w]$.

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 [18]. *G3K 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 *G3K* formulas. *G3K 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 *G3K*, which is proved to be sound and complete for the basic modal logic *K* [18].

CLASSICAL RULES

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

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: A prefixed tableau system for the modal logic K

2.3.4 Prefixed tableau systems

Prefixed tableaux 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 Free-variable prefixed tableau systems

Prefixed tableau systems have a deficiency that is also common in first-order sequent calculi. Resolution methods [21], which introduce meta-variables and unification may have an exponential speed-up in proof complexity over sequent calculi [2]. In a similar way, free-variable prefixed tableaux [20] aim at improving prefixed tableaux by the introduction of meta variables and simple unification. This construct allows for the delaying of the $\Box_F$ rule and might result with shorter proofs, as can be seen in Fig. 8 where the free-variable tableau has 9 rule applications versus the 12 of the standard tableau proof. The addition of meta-variables comes with the cost that careful restrictions must be posed on the tableau proofs in order to preserve soundness. In particular, the unification of these meta-variables must be restricted in order to prevent such unsoundness. In this paper, we are not interested in proof generation but in the structure of proofs only and will therefore omit further discussion on this topic. The interested reader can refer to [4] for further reading.

3 Certification of tableau modal proofs

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

In [17], 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 G3K 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 G3K. 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:

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

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 *G3K* sequents into *LKF* sequents:

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

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 [17], where a more formal statement and a detailed proof can be found.

Theorem 1 *Let Π be a G3K derivation of a sequent S from the sequents $S_1,\ldots,S_n$. Then there exists an LKF derivation Π' of $[S]$ from $[S_1],\ldots,[S_n]$, 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.*

Such a result is easily extended to the case of prefixed tableaux, by relying on the correspondence between prefixed tableaux and nested sequents [11], which are a subclass of labeled sequents. We omit the details.

3.2 Foundational proof certificate specifications

The translation presented in Section 3.1 can be used in order to check labeled sequent and prefixed tableau proofs in *LKF*. In fact, 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. We propose here two different FPC specifications. The first one requires a quite detailed proof evidence from the prover, while in the second one we only require the prover to provide some core information about the proof evidence and we check that it is correct by reconstructing the rest of the proof.

By observing G3K and prefixed tableau rules (Section 2.3), one can notice that a proof in these formalisms is fully represented by specifying:

1. at each step, on which formula we apply a rule;

2. in the case of a $\Diamond$-formula for G3K (or a $\Box$-formula for tableaux), with respect to which label (prefix) we apply the rule;
3. in the case of an initial (closure) rule, with respect to which complementary literal we apply it.

For this reason, an adequate and detailed proof evidence of a labeled sequent or prefixed tableau proof will consist in a tree describing the original proof (we will call it a *decide tree* in the following). 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 decide 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. Theorem 1 guarantees that at the end of a bipole, we will be in a situation which is equivalent to that of the corresponding G3K or tableau 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, in the proof evidence, the formula under consideration to the corresponding new-world-generating formula (a $\Box$-formula in the case of G3K; a $\Diamond$-formula in the case of tableaux). Similarly, in the case of a closure (3), the additional information in the node will specify the index of the complementary literal.

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 indexes must be provided: this is done in λ-Prolog by declaring constructors of the types `cert` and `index`. In Figure 4, we show the type declaration for the `fittings-tableaux` FPC that takes as input a decide tree corresponding to a proof, as specified above. In this declaration, we assume that `term` and `atm` are already declared, with the obvious intended meaning.

Several constructors are used to build indexes denoting formulas. `eind` is used to denote the root formula (ideally, the theorem to be proved). `lind` and `rind` are used to denote, respectively, the left and right direct subformulas of a formula (in case of a formula whose main connective is unary, we use `lind` to build the index of its only direct subformula). We have a specific constructor `bind` for subformulas of a $\Box$-formula in a tableau proof (or of a $\Diamond$-formula in a G3K proof). This takes two arguments, the first one being the index of the formula itself and the second one being the index of the corresponding eigenvariable-generating formula, i.e., of the formula that introduced the eigenvariable used in the current rule application. Finally, `none` is used in special cases when an index is not necessary. For instance, for the formula $((\Box p) \wedge (\Diamond \neg q)) \vee (\Box(\neg p \vee q))$ of Example 1 (Figure 8 (a)), we have the following (non exhaustive) mapping of indexes to formulas:

`eind` -> $1 : ((\Box p) \wedge (\Diamond \neg q)) \vee (\Box(\neg p \vee q))$
`lind(eind)` -> $1 : (\Box p) \wedge (\Diamond \neg q)$
`lind(lind(eind))` -> $1 : \Box p$
`rind(lind(eind))` -> $1 : \Diamond \neg q$
`bind(lind(lind(eind)),rind(lind(eind)))` -> $1.1 : p$

A `dectree` represents a decide tree as described above: the two indexes taken as arguments represent, respectively, the index of the formula on which to decide and the index possibly used for additional information; the third argument (a list of `dectrees`) is used to represent the (at most two) subtrees of this formula. Finally, we define a constructor `fitcert` that represents a certificate in the `fittings-tableaux` FPC specification. Together with the decide tree, representing the proof evidence, it takes two more arguments, which are used along the proof by the clerk and expert predicates: (i) a list of indexes to be used when storing formulas; and (ii) a list of pairs `(index, atm)` containing for each instantiated tableau $\Diamond$-formula the corresponding eigenvariable. Both such arguments are initialized as empty when a proof checking starts.

```
eind : index                              none : index
lind : index -> index                     rind : index -> index
bind : index -> index -> index

rel : term -> term -> atm

dectree : index -> index -> list dectree -> dectree
fitcert : list index -> dectree -> list (pair index atm) -> cert
```

Figure 4: Type declaration for the `fittings-tableaux` FPC specification.

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. In Figure 5, we define clerks and experts for the `fittings-tableaux` FPC specification. According to this specification, each decide step is completely determined by the proof evidence: in the $\texttt{decide}_e$ expert, `I` denotes the index of the formula on which to decide; notice also that the first element of the `fitcert` (the list of indexes) is reinitialized to be empty, as this list will be used, along the bipole, to construct appropriate indexes for the subformulas being created. For example, in $\texttt{orNeg}_c$, two indexes (`lind I` and `rind I`) are created and put inside such a list. In the end of the bipole, these indexes will be used to store the subformulas with a proper index. $\texttt{orNeg}_c$ (as well as $\texttt{andNeg}_c$) is described by two cases. The first one corresponds to the case when the $\vee^-$ rule ($\wedge^-$ rule) is being applied on the formula on which we have just decided; the second one corresponds to the case when the $\vee^-$ connective ($\wedge^-$ connective) arises from the translation of a $\Box$-formula ($\Diamond$-formula). In the former case, we have to take care of the indexes generated; in the latter, we do not. With regard to $\texttt{andPos}_e$, we remark that the connective $\wedge^+$ can only occur in a formula that is the translation of a $\Diamond$-formula and for this reason we have only one case. $\texttt{release}_c$ leaves things unchanged. In the case of $\texttt{all}_c$, we update the list of maps between formulas and corresponding eigenvariables and in $\texttt{some}_e$, by using the information contained in such a list and the information contained in the second index of the current node in the decide tree (specifying to which universal formula the current existential formula corresponds), we are able to instantiate the $\exists$ with the proper eigenvariable. In the $\texttt{store}_c$ clerk, we use the index created so far to properly store the formula under consideration; note that in the case of relational atoms, we simply store the formula with the index `none`. Finally, in order to apply an `initial` rule, the expert $\texttt{initial}_e$ checks that the complementary formula is indexed with the second element of the decide tree (`O` in the specification).

As already remarked, the `fittings-tableaux` FPC specification allows for a very detailed and completely faithful checking of an original proof in both the G3K sequent calculus and the Fitting-style prefixed tableau system of Section 2.3.4. In fact, at least as long as the logic K is considered, a proof of a modal formula A in the first setting can be easily converted into a refutation of $\neg A$ in the second formalism, where: at a given connective rule in G3K corresponds the rule of the dual connective in a tableau and at an initial rule application corresponds a closure of a branch[1]. This means that we can use the same format for proof evidence, the same translation and the same FPC specification for both the formalisms. In the case of a theorem proved in G3K, we will obtain in *LKF* a proof of its translation (as defined in Section 3.1); in the case of prefixed tableaux, we will obtain a proof of the translation of the negated formula. Further remarks on the implementation and on some experiments will be given in

[1]Clearly, if one considers the two-sided sequent version of G3K given here, it is also necessary to take care of the different meaning of having a connective on the left or on the right side of the sequent. This can be done by defining a translation from two-sided G3K sequents into one-sided sequents, as shown, e.g., in [17].

```
∀ L,I,O,D,M. decide_e (fitcert L (dectree I O D) M) I (fitcert [] (dectree I O D) M).

∀ I,O,H,T,M. orNeg_c (fitcert [] (dectree I O [H|T]) M) (fitcert [lind I, rind I] H M).
∀ E,L,D,M. orNeg_c (fitcert [E|L] D M) (fitcert [E|L] D M).

∀ I,O,H,G,T,M. andNeg_c (fitcert [] (dectree I O [H,G|T]) M) (fitcert [lind I] H M) (
    fitcert [rind I] G M).
∀ E,L,D,M. andNeg_c (fitcert [E|L] D M) (fitcert [E|L] D M) (fitcert [E|L] D M).

∀ L,I,O,T,M,F. andPos_e (fitcert L (dectree I O T) M) (fitcert L (dectree I none T) M) (
    fitcert L (dectree I O T) M).

∀ C. release_e C C.

∀ I,O,H,T,M. all_c (fitcert [] (dectree I O [H|T]) M) (Eigen\ fitcert [lind I] H [pair I
    Eigen|M]).

∀ I,O,H,T,M,X. some_e (fitcert [] (dectree I O [H|T]) M) X (fitcert [bind I O] H M) :- (
    pair O X) ∈ M.

∀ C,X,Y. store_c C (n (rel X Y)) none C :- !.
∀ C,X,Y. store_c C (p (rel X Y)) none C :- !.
∀ H,T,D,M,F. store_c (fitcert [H|T] D M) F H (fitcert T D M).

∀ L,I,O,D,M. initial_e (fitcert L (dectree I O D) M) O.
```

Figure 5: Definition of the `fittings-tableaux` FPC specification.

Section 3.3

Having such a faithful proof representation can appear in some cases rather naive and space-consuming. It is quite common, in the context of proof checking, to work with less precise proof evidences, that only contain crucial information about a given proof, and let the checker perform some proof reconstruction of the rest. The second FPC specification that we propose, `simpfit-tableaux` (Figures 6 and 7), aims at this goal. Instead of telling LKF, at each beginning of a new bipole, on which formula to decide, we only provide the checker with the following information:

- a mapping between $\Box$ and $\Diamond$-formulas, i.e., which eigenvariable to use at each instantiation of an existential formula;
- a mapping between complementary literals, i.e., all the pairs of literals with respect to which we can apply an `initial` rule.

This information is specified in the proof evidence by two further constructors: `boxinfo` and `closure`. Together with such a proof evidence, a `simpfit-tableaux` certificate requires other elements:

- an integer, used as a flag, for dealing with the creation of indexes: 1 means that we have just decided on a formula and thus we need to create subindexes at the first rule application concerning a connective; we have 0 otherwise;
- a list of indexes, used, as in `fittings-tableaux`, for storing formulas with the proper index;
- a list of maps between universal formulas and eigenvariables, updated by the $\texttt{all}_c$ clerk;
- a further list of indexes denoting formulas on which we have already decided: since the decide step is not driven by the certificate, as in the previous FPC, this list is required in order to avoid that the checker keeps deciding on a same formula.

```
    boxinfo : index -> index -> boxinfo.
    closure : index -> index -> closure.
    use : index -> use.

    simpfitcert : int -> list index -> list closure -> list boxinfo -> list (pair
index atm) -> list use -> cert.
```

Figure 6: Type declaration for the `simpfit-tableaux` FPC specification.

The `simpfit-tableaux` FPC specification is rather general and can be used to certify proofs also in formalisms that are farther from sequent calculus (and from *LKF* in particular). For example, we applied this FPC to check proofs given in the free-variable tableau systems of Section 2.3.5. A faithful reproduction of a proof in such a formalism would not be trivial in *LKF*, because free-variable tableaux admit the application, on a given branch, of a $\Box$-rule before the corresponding $\Diamond$-rule (see, e.g., tableau (b) in Figure 8). However, we can extract from such proofs the information about eigenvariable instantiation and closures and use it while freely constructing the rest of the proof.

3.3 Examples

In this section we apply the specifications from the previous section to several examples. Since none of the theorem provers we have experimented with produced a proof as an output, we had to modify their source codes in order to obtain some information about their execution states and then create the proof evidences by hand. The prover we have chosen to produce this partial information with is ModLeanTAP [4], a free variable modal tableau prover written in Prolog. Here we consider the following two examples:

$$1.\ (\Box(p \Rightarrow q)) \Rightarrow ((\Box p) \Rightarrow (\Box q)) \qquad\qquad 2.\ (\Diamond\neg p \vee \Diamond\neg q) \wedge \Box(p \wedge q)$$

The first example is taken from the ModLeanTAP self testing benchmark [2]. The second example is more involved since it requires the setting of two different worlds for the same box formula. This example is taken from [4] and demonstrates the importance of an FPC specification for the proofs generated by free-variable prefixed-tableaux as can be seen from Fig. 8. This figure shows the prefixed tableaux and free-variable prefixed tableaux derivations of the two examples.

We have extracted two types of proofs for the above examples using ModLeadTAP. The first type of proofs contains detailed step-by-step evidence. This form of proof can normally be extracted from prefixed tableaux and G3K proofs. The second type contains only the essential information: the relationship between all the boxes and diamonds and between all literal which were used for closing a branch. Since free-variable tableau provers often use optimization techniques which breaks the relationship between their proof structure and that of normal tableaux, most of this information loses its importance and we require only the essential information to be included.

Checkers can be obtained online [3] and can be executed by running in a bash terminal: [4].

```
$ ./prover.sh ftab1
```

where the argument is the name of the λProlog module denoting the proof evidence one wishes to check. In this example, the name is the module of the detailed proof (to be used with the `fittings-tableaux`

[2] A set of problems which can be found in modleantest.pl in `http://formal.iti.kit.edu/beckert/modlean/`.

[3] The exact version can be found on the "gandalf2016" branch in the git repository `https://github.com/proofcert/checkers`.

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

```
∀ F,I,C,B,E,U,D,U2. decide_e (simpfitcert F I C B E U) D (simpfitcert 1 [D] C B E U2) :-
    use D ∈ U, U2 = U \ {use D}.
∀ F,I,C,B,E,U. decide_e (simpfitcert F I C B E U) none (simpfitcert 1 [none] C B E U).

∀ I,C,B,E,U. orNeg_c (simpfitcert 1 [I] C B E U) (simpfitcert 0 [lind I, rind I] C B E U).
∀ I,C,B,E,U. orNeg_c (simpfitcert 0 I C B E U) (simpfitcert 0 I C B E U).

∀ I,C,B,E,U. andNeg_c (simpfitcert 1 [I] C B E U) (simpfitcert 0 [lind I] C B E U) (
    simpfitcert 0 [rind I] C B E U).
∀ I,C,B,E,U. andNeg_c (simpfitcert 0 I C B E U) (simpfitcert 0 I C B E U) (simpfitcert 0 I
    C B E U).

∀ F,I,C,B,E,U. andPos_e (simpfitcert F I C B E U) (simpfitcert 0 I C B E U) (simpfitcert 0
    I C B E U).

∀ C. release_e C C.

∀ F,I,C,B,E,U. all_c (simpfitcert F [I] C B E U) (Eigen\ simpfitcert 0 [lind I] C B [pr I
    Eigen|E] U).

∀ F,I,C,B,E,U,X. some_e (simpfitcert F [I] C B E U) X (simpfitcert 0 [bind I 0] C B2 E [(
    use I)|U]) :- (pair 0 X) ∈ E, (boxinfo I 0)∈B, B\{boxinfo I 0}=B2.

∀ F,I,C,B,E,U,X,Y. store_c (simpfitcert F I C B E U) (n (rel X Y)) none (simpfitcert 0 I C
    B E U) :- !.
∀ F,I,C,B,E,U,X,Y. store_c (simpfitcert F I C B E U) (p (rel X Y)) none (simpfitcert 0 I C
    B E U) :- !.
∀ F,H,T,C,B,E,U,A. store_c (simpfitcert F [H|T] C B E U) (n A) H (simpfitcert 0 T C B E U)
    :- !.
∀ F,H,T,C,B,E,U,Form. store_c (simpfitcert F [H|T] C B E U) Form H (simpfitcert 0 T C B E
    [(use H)|U]).

∀ F,I,T,C,B,E,U,Compl. initial_e (simpfitcert F [I|T] C B E U) Compl :- member (closure I
    Compl) C.
∀ F,I,T,C,B,E,U,Compl. initial_e (simpfitcert F [I|T] C B E U) Compl :- member (closure
    Compl I) C.
∀ F,I,T,C,B,E,U. initial_e (simpfitcert F [I|T] C B E U) none.
```

Figure 7: Definition of the `simpfit-tableaux` FPC specification.

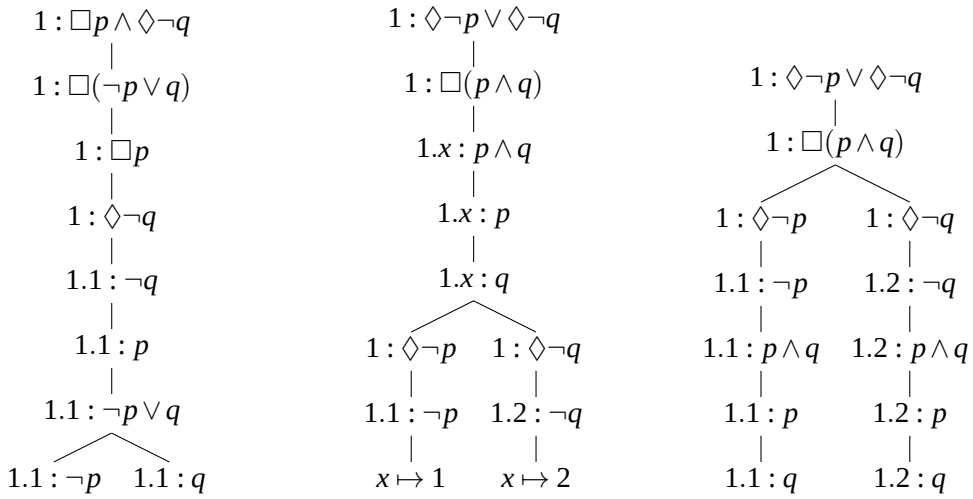
FPC specification) of the first problem, which can be found in Fig. 9. The module for the "essential" one (to be used with the `simpfit-tableaux` FPC specification) can be found in Fig. 10. More examples can be found in the following folder (shipped together with Checkers):

`/src/test/tableaux`

4 Conclusion

The examples given in the previous section were relatively simple and it is our intention to apply the tool to a real benchmark of problems, such as the one described in [3]. In order to achieve this goal, we need to find tableau provers which output detailed enough proofs.

We also aim at extending the approach to variants of the logic K. With regard to this, we remark that the translation of [17] for G3K sequents was proved to be effective not only for K but for all those modal logics characterized by Kripke frames whose relational properties can be expressed by means of geometric theories (most common modal logics fall within this class). Extending our approach to deal with such logics in the case of labeled sequent calculi is therefore straightforward. If we consider



(a) Tableau for example 1 (b) FV tableau for example 2 (c) Tableau for example 2

Figure 8: Prefixed and free-variable prefixed tableau derivations

```
module ftab1.                                                    % module declaration
accumulate fittings-tableaux.                                    % fpc specification module
accumulate lkf-kernel.                                           % kernel module
modalProblem "Detailed_proof_of_ModLeanTAP_problem_t1"           % problem description
(((dia (-- p1)) !! (box (++ q1))) !! (dia ((++ p1) && (-- q1)))) % modal theorem
(fitcert [] (                                                    % proof evidence
 (dectree eind none [
  (dectree (lind eind) none [
   (dectree (rind (lind eind)) none [
    (dectree (lind (lind eind)) (rind (lind eind)) [
     (dectree (rind eind) (rind (lind eind)) [
      (dectree (bind ((rind eind)) ((rind (lind eind)))) none [
       (dectree (lind (bind ((rind eind)) ((rind (lind eind))))) (bind ((lind (lind eind))) ((rind (lind eind)))) []),
       (dectree (lind (rind (lind eind))) (rind (bind ((rind eind)) ((rind (lind eind))))) [])])])])])])])) [] ).
```

Figure 9: src/test/tableaux/ftab1.mod

Fitting-style prefixed tableaux, however, we notice that they are typically extended to capture specific relational properties, e.g., transitivity, not by using rules that operate on the relational atoms, but rather by modifying the existing rules concerning modalities or by adding further such rules. This tends to break the strict correspondence between rule applications and bipoles, but, as observed in [14], such a correspondence can be somehow recovered, e.g., by using an encoding that involves the application of a cut rule.

In the spirit of generality pursued by the ProofCert project, we also plan to consider deductive formalisms other than tableaux, e.g., resolution. Since propositional modal resolution provers are often based on a translation into a first-order language, we expect to be able to reuse part of this work also in that setting.

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

References

[1] Jean-Marc Andreoli (1992): *Logic Programming with Focusing Proofs in Linear Logic*. *J. Log. Comput.* 2(3), pp. 297–347, doi:10.1093/logcom/2.3.297.

```
module sftab1.
accumulate simpfit-tableaux.
accumulate lkf-kernel.
modalProblem "Essential proof of ModLeanTAP problem t1"
(((dia (-- p1)) !! (box (++ q1))) !! (dia ((++ p1) && (-- q1))))
(simpfitcert 1 [eind]
 [ closure (lind (bind (rind eind) (rind (lind eind)))) (bind (lind (lind eind)) (rind (lind eind))),
   closure (lind (rind (lind eind))) (rind (bind (rind eind) (rind (lind eind)))) ]
 [ boxinfo (lind (lind eind)) (rind (lind eind)),
   boxinfo (rind eind) (rind (lind eind)) ] [] [] ).
```

Figure 10: src/test/tableaux/sftab1.mod

[2] Matthias Baaz & Alexander Leitsch (1992): *Complexity of Resolution Proofs and Function Introduction*. *Ann. Pure Appl. Logic* 57(3), pp. 181–215, doi:10.1016/0168-0072(92)90042-X.

[3] Peter Balsiger, Alain Heuerding & Stefan Schwendimann (2000): *A Benchmark Method for the Propositional Modal Logics K, KT, S4*. *J. Autom. Reasoning* 24(3), pp. 297–317, doi:10.1023/A:1006249507577.

[4] Bernhard Beckert & Rajeev Goré (2001): *Free-Variable Tableaux for Propositional Modal Logics*. *Studia Logica* 69(1), pp. 59–96, doi:10.1023/A:1013886427723.

[5] Patrick Blackburn & Johan Van Benthem (2007): *Modal logic: a Semantic Perspective*. In Frank Wolter Patrick Blackburn, Johan van Benthem, editor: *Handbook of Modal Logic*, Elsevier, pp. 1–82, doi:10.1016/S1570-2464(07)80004-8.

[6] Mathieu Boespflug, Quentin Carbonneaux & Olivier Hermant (2012): *The λΠ-calculus modulo as a universal proof language*. In: *the Second International Workshop on Proof Exchange for Theorem Proving (PxTP 2012)*, 878, pp. pp–28.

[7] Raphaël Cauderlier & Pierre Halmagrand (2015): *Checking Zenon Modulo Proofs in Dedukti*. In Cezary Kaliszyk & Andrei Paskevich, editors: *Proceedings Fourth Workshop on Proof eXchange for Theorem Proving, PxTP 2015, Berlin, Germany, August 2-3, 2015.*, *EPTCS* 186, pp. 57–73, doi:10.4204/EPTCS.186.7.

[8] Zakaria Chihani, Tomer Libal & Giselle Reis (2015): *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*, *Lecture Notes in Computer Science* 9323, Springer, pp. 201–210, doi:10.1007/978-3-319-24312-2_14.

[9] Melvin Fitting (1972): *Tableau methods of proof for modal logics*. *Notre Dame Journal of Formal Logic* 13(2), pp. 237–247, doi:10.1305/ndjfl/1093894722.

[10] Melvin Fitting (2007): *Modal proof theory*. In Patrick Blackburn, Johan van Benthem & Frank Wolter, editors: *Handbook of Modal Logic*, Elsevier, pp. 85–138, doi:10.1016/S1570-2464(07)80005-X.

[11] Melvin Fitting (2012): *Prefixed tableaus and nested sequents*. *Ann. Pure Appl. Logic* 163(3), pp. 291–313, doi:10.1016/j.apal.2011.09.004.

[12] Dov M. Gabbay (1996): *Labelled Deductive Systems*. Clarendon Press.

[13] Chuck Liang & Dale Miller (2009): *Focusing and polarization in linear, intuitionistic, and classical logics*. *Theor. Comput. Sci.* 410(46), pp. 4747–4768, doi:10.1016/j.tcs.2009.07.041.

[14] Sonia Marin, Dale Miller & Marco Volpe (2016): *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*. To appear.

[15] Dale Miller (2011): *ProofCert: Broad Spectrum Proof Certificates*. An ERC Advanced Grant funded for the five years 2012-2016.

[16] Dale Miller & Gopalan Nadathur (2012): *Programming with Higher-Order Logic*. Cambridge University Press, doi:10.1017/CBO9781139021326. Available at http://www.cambridge.org/

de/academic/subjects/computer-science/programming-languages-and-applied-logic/programming-higher-order-logic?format=HB.

[17] Dale Miller & Marco Volpe (2015): *Focused Labeled Proof Systems for Modal Logic*. In Martin Davis, Ansgar Fehnker, Annabelle McIver & Andrei Voronkov, editors: *Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings*, *Lecture Notes in Computer Science* 9450, Springer, pp. 266–280, doi:10.1007/978-3-662-48899-7_19.

[18] Sara Negri (2005): *Proof Analysis in Modal Logic*. *J. Philosophical Logic* 34(5-6), pp. 507–544, doi:10.1007/s10992-005-2267-3.

[19] Hans Jürgen Ohlbach (1988): *A Resolution Calculus for Modal Logics*. In Ewing L. Lusk & Ross A. Overbeek, editors: *9th International Conference on Automated Deduction, Argonne, Illinois, USA, May 23-26, 1988, Proceedings*, *Lecture Notes in Computer Science* 310, Springer, pp. 500–516, doi:10.1007/BFb0012852.

[20] Steve Reeves (1987): *Semantic Tableaux As a Framework for Automated Theorem-proving*. In: *On Advances in Artificial Intelligence*, John Wiley & Sons, Inc., New York, NY, USA, pp. 125–139. Available at http://dl.acm.org/citation.cfm?id=29992.30001.

[21] John Alan Robinson (1965): *A Machine-Oriented Logic Based on the Resolution Principle*. *J. ACM* 12(1), pp. 23–41, doi:10.1145/321250.321253.