# Certification of labeled proofs for modal logics with geometric frame conditions

Tomer Libal, Marco Volpe

▶ **To cite this version:**

Tomer Libal, Marco Volpe. Certification of labeled proofs for modal logics with geometric frame conditions. 2017. hal-01643120

**HAL Id: hal-01643120**
**https://hal.science/hal-01643120v1**

Preprint submitted on 21 Nov 2017

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Certification of labeled proofs for modal logics with geometric frame conditions

Tomer Libal, Marco Volpe
INRIA
Technical report

June, 8th 2017

**Abstract**

Several proof formalisms have been used, and in some cases even introduced, in order to define proof systems for modal logic. Our work falls within a more general project of establishing a common specification language for checking proofs given in a wide range of deductive formalisms. In this paper, we consider the case of labeled proof systems for modal logics, i.e., in particular, Negri's labeled sequent calculi, Fitting's prefixed tableaux and free-variable prefixed tableaux, and provide a framework for certifying proofs given in such calculi. The method is based on the use of a translation from the modal language into a first-order polarized language and on a checker whose trusted kernel is a simple implementation of a classical focused sequent calculus. The framework allows for a high flexibility in the representation of proofs to be checked, in the sense that even partial proofs can be verified by employing a process of proof reconstruction. We describe the general method for modal logics characterized by geometric frame conditions, present its implementation in a Prolog-like language, and provide several examples of proof certification in the case of well-known normal modal logics, like K, S4 and S5.

# 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 [1] and the resolution principle [2]. 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.

The main goal of this paper is to try and define a framework in which arbitrary results obtained by modal theorem provers can be certified. Such a framework can, therefore, bridge the gap between optimization and trust - theorem provers need not be proven correct but must produce "sufficient" proof evidence which can be then certified. A proposal for a definition of what is considered as a sufficient proof evidence is one of the topics we address, as well as a description of the framework, its implementation and its usage.

ProofCert [3] is a project targeting 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 [4] 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 $LKF$ [5]. In [6], a translation from the language of the labeled sequent system $G3K$ [7] 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 [8], we proposed two different FPC specifications for tableau and labeled sequent proofs for the modal logic K. The first one required quite a

detailed proof evidence from the prover, while in the second one we only required 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 contrast, in this paper we extend the result in three directions.

First, we extend the required detail level of the proofs to the whole spectrum between the two mentioned above. Proofs can now omit any amount of detail as long as the remaining information is enough for adequate certification, thus emphasizing a trade-off between proof checking adequacy and proof search: while a high amount of details allows for an adequate and precise certification of the original proof, a proof representation with holes will require a certain degree of proof search in order to deduce the missing components. This flexibility allows, for example, certification of proofs given in the formalism of free variable tableaux, for which a step-by-step proof checking by means of emulation in a sequent calculus would not be possible.

Second, we add support for proofs in both validation-style labeled sequent systems and refutation-style prefixed tableau systems. While such a support for the logic K was already existing in [8], it was based on a simple translation. The current approach is based on two distinct FPC specifications. The first supports the $G3K$ calculus while the second supports tableau systems.

Last, we add support for proofs of any modal logic whose semantical frames can be defined by geometric properties and not only for the logic K, thus allowing the certification of proofs from modal logics such as S4 and S5.

Proof evidences arising from the labeled proof systems mentioned above, 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 the different formalisms, by making use of the same notion of 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 [9] independent certification for the classical first-order tableau prover Zenon modulo [10].

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 $\lambda$Prolog [11] 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 $\lambda$Prolog description of a proof output of a theorem prover. Given the high-level declarative form of $\lambda$Prolog, 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 $\lambda$Prolog 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 [12] 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 [5]. 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 $\Xi$. At the same time, each rule is associated with a predicate (for example $\mathit{initial}_e(\Xi, l)$) 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 Labeled proof systems 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 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 $V : W \to 2^{\mathcal{P}}$ is a function that assigns to each world in $W$ a (possibly empty) set of propositional symbols.

| Axiom | Condition | First-Order Formula |
|---|---|---|
| T: $\Box A \supset A$ | Reflexivity | $\forall w.R(w,w)$ |
| 4: $\Box A \supset \Box\Box A$ | Transitivity | $\forall w,v,u.(R(w,v) \wedge R(v,u)) \supset R(w,u)$ |
| B: $A \supset \Box\Diamond A$ | Symmetry | $\forall w,v.R(w,v) \supset R(v,w)$ |
| 5: $\Diamond A \supset \Box\Diamond A$ | Euclideaness | $\forall w,v,u.(R(w,v) \wedge R(w,u)) \supset R(v,u)$ |

Table 1: Axioms and corresponding first-order conditions on $R$.

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

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

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 $\mathcal{M}$.

The former definition characterizes the basic modal logic $K$. Several further modal logics can be defined as extensions of $K$ by simply restricting the class of frames we consider. Many of the restrictions we are interested in are definable as formulas of first-order logic where a binary predicate $R(w, w')$ refers to the corresponding accessibility relation. Table 1 summarizes some of the most common modal logics, describing the corresponding frame property, together with the modal axiom capturing it [13]. We will refer to the logic satisfying the axioms $F_1, \ldots, F_n$ as $KF_1 \ldots F_n$. In the literature, some of these modal logics are referred to with specific names, e.g., $S4$ denotes the logic $KT4$ and $S5$ the logic $KTB4$ or, equivalently, $KT5$. We will sometimes use these names in the following.

### 2.3.2 The standard translation from modal logic into classical logic

The following *standard translation* (see, e.g., [14]) provides a bridge between propositional 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* [14] 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 calculi

Several different deductive formalisms have been used for modal proof theory and theorem proving. One of the most interesting approaches has been presented in [15] 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 [7] for the modal logic $K$. In this paper will refer to it as $LS_K$. *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 $\varphi, \psi$ to denote $LS$ formulas. *LS sequents* have the form $\Gamma \vdash \Delta$, where $\Gamma$ and $\Delta$ are multisets containing labeled formulas and relational atoms. In Fig. 2, we present the rules of $LS_K$, which is proved to be sound and complete for the basic modal logic $K$ [7].

Then in Figure 3, we present the rules for reflexivity, symmetry and transitivity. By adding, modularly, one or more of such rules one can obtain a system for the modal logic defined by the corresponding axioms. We will denote by $LS_{KR_1 \ldots R_n}$ any system obtained by adding to $LS_K$ the rules $R_1, \ldots, R_n \in \{T_{LS}, B_{LS}, 4_{LS}\}$. For simplicity, we will also write $LS_{S4}$ to denote the system $LS_{KT4}$ and $LS_{S5}$ for $LS_{KTB4}$. Finally, we will write $LS$ to refer to a generic proof system in this class, when we do not need to specify the specific logic we want to capture. From [7], we know that any proof system $LS_{KR_1 \ldots R_n}$ is sound and complete for the logic $KF_1 \ldots F_n$.

All along the paper, as a running example, we will use derivations of the axiom 5 for the euclideaness property. In Figure 4, we present a derivation

of such an axiom in the proof system $LS_{S5}$.

### 2.3.4 Prefixed tableau systems

Prefixed tableaux can also be seen as a particular kind of labeled deductive system. They have been introduced in [1]. The formulation that we use here is closer to the one in [16] 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 $\sigma$ is a prefix, then $\sigma.1$ and $\sigma.2$ denote two worlds accessible from $\sigma$. A *prefixed formula* is $\sigma : A$, where $\sigma$ 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 5. We say that a branch of a tableau is a *closed branch* if it contains $\sigma : P$ and $\sigma : \neg P$ for some $\sigma$ 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 5 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 $\sigma$, 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 $\sigma$. We say that a prefix is *used* on a branch if it already occurs in the tableau branch and it is *new* otherwise.

The system of Figure 5, to which we give the name of $PT_K$, is sound and complete for the logic $K$. In Figure 6, we present rules [16] for capturing some variants of $K$. As for labeled sequent systems, tableau systems for variants of $K$ can be defined modularly by adding one or more of such rules. There is an exception though. We need a further rule, which reads as a combination of symmetry and transitivity, in order to get a system for the logic $S5$. In Table 2, we show how to use the rules of Figure 6 in order to generate the tableau systems that will be used in the rest of the paper.

In Figure 7, we present a derivation of the axiom 5 in the system $PT_{S5}$.

### 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 [17], which introduce meta-variables and unification may have an exponential speed-up in proof complexity over

| Logic | System | Special rules |
|---|---|---|
| $KT$ | $PT_{KT}$ | $T_{PT}$ |
| $K4$ | $PT_{K4}$ | $4_{PT}$ |
| $KB$ | $PT_{KB}$ | $B_{PT}$ |
| $S4$ | $PT_{S4}$ | $T_{PT}$, $4_{PT}$ |
| $S5$ | $PT_{S5}$ | $T_{PT}$, $4_{PT}$, $4r_{PT}$ |

Table 2: Prefixed tableau systems.

sequent calculi [18]. In a similar way, free-variable prefixed tableaux [19] aim at improving prefixed tableaux by the introduction of meta variables and simple unification. This construct allows for the delaying of the $\Box_{PT}$ rule and might result with shorter proofs, as can be seen in Fig. 8 where the free-variable tableau for the formula $(\Diamond\neg p \vee \Diamond\neg q) \wedge \Box(p \wedge q)$ 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 [20] for further reading.

# 3 Certification of modal proofs

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

In [6], 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 an $LS$ proof of $A$ and bipoles in an $LKF$ proof of $A'$ holds. Such a correspondence has been used in order to prove an 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 $A^{\partial^+}$, 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 $LS$ sequents into $LKF$ sequents:

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

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

Application of relational rules in extensions of $LS_K$ correspond to instantiations of the corresponding axioms in $LKF$. We recall that a geometric axiom has the form:

$$\forall \overline{z}(P_1 \wedge \ldots \wedge P_m \supset (\exists x_1(Q_{11} \wedge \ldots \wedge Q_{1k_1}) \vee \ldots \vee \exists x_n(Q_{n1} \wedge \ldots \wedge Q_{nk_n})))$$

In $LKF$, we can proceed by adding such axioms in the left-side of the sequent to be derived. We propose the following translation, involving polarization of connectives, for an axiom having the form shown above[1]:

$$\exists \overline{z}((P_1 \wedge^+ \ldots \wedge^+ P_m) \wedge^+ (\forall x_1(\neg Q_{11} \vee^- \ldots \vee^- \neg Q_{1k_1}) \wedge^- \ldots \wedge^- \forall x_n(\neg Q_{n1} \vee^- \ldots \vee^- \neg Q_{nk_n})))$$

[1]Note that in $LKF$ we consider one-sided sequents and the one we propose is in fact a polarization of the negation of the axiom.

We have that each instantiation of such a polarized axiom corresponds to a single bipole in *LKF*.

We recall here a result from [6], where a more formal statement and a detailed proof can be found.

**Theorem 1** *Let* $\Pi$ *be an LS derivation of a sequent* $S$ *from the sequents* $S_1, \ldots, S_n$*. Then there exists an LKF derivation* $\Pi'$ *of* $[S]$ *from* $[S_1], \ldots, [S_n]$*, such that each rule application in* $\Pi$ *corresponds to a bipole in* $\Pi'$*. 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 [21], which are a subclass of labeled sequents. In the case of some relational rules for prefixed tableaux, i.e. transitivity, a proof of adequacy can sometimes require a given number of intermediate steps in order to recover the correspondence between the original proof and the one in *LKF*, similarly to what has been done in [22]. We omit the details here.

## 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.

In this section we will first define our notion of a sufficient proof evidence and will then describe the three extensions to the framework in [8] which were discussed in the introduction.

### 3.2.1 The proof evidence

By observing *LS* and *PT* 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 *LS* (or a $\Box$-formula for *PT*), 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;

4. when using an axiom corresponding to a relational rule such as transitivity, its specific instantiation.

For this reason, an adequate and detailed proof evidence of a labeled sequent or prefixed tableau proof will consist in a tree describing the principal formulas we apply rules to (we will call it a *decide tree* in the following) and an additional data structure for specifying the referenced label (prefix) of a $\Diamond$-formula ($\Box$-formula), the complementary literal for the initial (closure) rule and the instantiations for the axioms used. The second data structure will be called the *essential map*, where in our notation, maps extend functions and refer to sets of pairs of values.

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 *LS* or *PT* 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 *LS*; a $\Diamond$-formula in the case of *PT*). Similarly, in the case of a closure (3), the additional information will specify the index of the complementary literal. Axiom instantiation (4) is similar to (2) and contain the indices of the formulas which generate the referred-to worlds. The decide tree will contain the steps in item (1) while items (2), (3) and (4) will be part of the essential map.

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 $\lambda$-Prolog by declaring constructors of the types `cert` and `index`. In Figure 9, we show a part of the type declaration for the `LS` FPC that takes as input a decide tree and an essential map 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 indices 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 an $LS$ 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. The intuition behind this index is to account for contractions, which take a simplified form in modal logic proofs - first they correspond to instantiations of $\Box$-formulas in a tableau proof ($\Diamond$-formulas in an $LS$ proof) but at the same time, these instantiations correspond to eigenvariables which are introduced by $\Diamond$-formulas ($\Box$-formulas). Finally, `relind` is a general index which refers to the relations appearing in the proof.

For instance, the root formula - $x : \Box\neg p \vee \Box\Diamond p$ - is denoted using the root index `eind`. Since the top symbol of the root formula is $\vee$, we refer to each disjunct using the left and right indices `lind(eind)` and `rind(eind)`. Subformulas of unary connectives like $\neg, \Box$ and $\Diamond$ are denoted using the left index.

Using this indexing mechanism and given the $LS_{S5}$ proof from Fig. 4, Fig. 10 shows the decide tree corresponding to the proof. As can be seen, the decide tree just contains either the indices of the principle formulas of the original proof or the actual rule names in the case of axioms. In the later case, the rule names uniquely determine the axiom formulas we need to decide on.

As mentioned before, the essential map contains information on how to apply the $\Diamond$ rule and the axioms as well as the complementary literals for the init rules. This information is essential for an adequate proof checking and is part of any proof.

An interesting technical problem is how to denote the information about which labels to choose when applying $\Box$ rules ($\Diamond$ in labeled proofs). These labels correspond to new labels which were introduced earlier by $\Diamond$ (respectively $\Box$) but, being fresh (referred to as eigenvariables later), are unknown to the proof evidence. Our solution is to denote these labels by referring to the indices of the $\Diamond$ (respectively $\Box$) formulas which introduced them.

| Map name | Index list |
|---|---|
| diabox | (lind (rind eind)) , (lind eind) |
| closure | (bind (lind (rind eind)) (lind eind)) , (lind (lind eind)) |
| symmetry instantiation | initial, (rind eind) |
| transitivity instantiation | (rind eind), initial, (lind eind) |

Table 3: The essential map of the certificate for the proof in Fig. 4.

The essential map corresponding to the above proof can be found in Table 3

The full certificate is given in Fig. 11.

### 3.2.2 The FPC specifications

As discussed in section 2.1, the general checker requires, in addition to a certificate, a program which allows the kernel to interpret a certificate. This program, called an FPC specification, contains logical definitions of the clerks and expert predicates. Each of the connectives in the kernel is being guided by such clerks and experts. Writing no specification for a given predicate defines that predicate to hold for no list of arguments and therefore, prevents the kernel from processing this connective. Similarly, writing a specification for a certain list of arguments restricts the kernel to these arguments only. In Figure 12, we define clerks and experts for the `LS` FPC specification.

According to this specification, each decide step is completely determined by the proof evidence: in the $\texttt{decide}_e$ expert, the variable `I` denotes the index of the formula on which to decide. For example, in $\texttt{orNeg}_c$, two indices (`lind I` and `rind I`) are created and put inside some list. In the end of the bipole, these indices will be used to store the subformulas with a proper index. It is important to note, that this list is not part of the certificate and is used just to manipulate and store intermediary information which is required for the correct execution of `checkers`. We call this list and other such data structures the *state*. Since the implementation of the state is a technical detail which is not required for the understanding of how we certify proofs, we will mention it only when necessary for the understanding of the rest of the program.

$\texttt{orNeg}_c$ is described by four cases. The first one corresponds to the case when the $\vee^-$ rule is being applied on the formula on which we have just decided; the rest correspond to the case when the $\vee^-$ connective arises from

the translation of a $\Box$-formula. It might be that there is a relational atom on the left side, in which case we need to tell the kernel to store it with the generic index `relind`.

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 two cases, similar to the three cases above.

$\texttt{release}_c$ leaves things unchanged.

In the case of $\texttt{all}_c$, we just need to link the index of the formula to the generated eigenvariable. These links are also part of the program state.

The $\texttt{some}_e$ expert is among the most sophisticated ones. It takes into account two components of the essential map, the correspondence between modalities as well as the instantiations of the axioms. The former is being taken care of by the first case while the latter is being treated by the other two. Part of the information necessary, the link between indices and eigenvariables, is stored in the state.

The definition of this expert also demonstrates how we have chosen to treat modalities other than K. A proof evidence which assumes a background modal theory, such as S4, must include the axioms of the theory as part of the certificate. The second change to the proof evidence is the addition of an axiom instantiation list as part of the essential map.

`checkers` is responsible for using the first list for storing the axioms. The application of an axiom is now stated in the decide list, and the specific instantiation, in the essential map.

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 `relind`. In between of the creation of the index and the actual call for store, we keep the information in the state.

Finally, in order to apply an `initial` rule, the expert $\texttt{initial}_e$ checks that the complementary formula is chosen properly using information from the essential map.

As already remarked, the above FPC specification allows for a very detailed and completely faithful checking of an original proof in the $LS$ sequent calculus. The FPC specification for $PT$ proofs contains many similarities to the one for $LS$. 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: a given connective rule in $LS$ corresponds to the rule of the dual connective in a $PT$ and an initial rule

application corresponds to a closure of a branch[2]. When considering modal logics other than K, the presence of axioms and their different treatment in the different proof systems cause the two FPC specifications to diverge a bit. The main difference lies in the instantiations of the axioms and requires some more information to be stored in the state. More precisely, the FPC specification for $LS$ is closely related to classical sequent calculus while the one for $PT$ differs more significantly. This is most obvious in the way axioms are applied. E.g., in $LS$, the application of the rule $4_{LS}$ corresponds to instantiating the axiom for transitivity in the classical sequent calculus. This is not the case in $PT$, where an application of the rule $4_{PT}$ corresponds to two rule applications: $4_{PT}$ plus $\Box_{PT}$. In order to capture this behavior, we have chosen to use a tactic language. Using this language, such a step in $PT$ is being denoted by a term which contains the relevant information for both rule applications in sequent calculus. We will not discuss further the implementation of the $PT$ FPC specification but point the interested reader to the $\lambda$Prolog implementation[3].

Further remarks on the implementation and on some experiments will be given in 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 approach we have chosen is to take advantage of one of Prolog features in order to support an arbitrary level of abstraction in the proof evidence - using anonymous variables, the proof evidence can choose to omit any piece of information, letting the certifier reconstruct the missing parts. One should note though, that abstracting over the essential map might result in the certifier being fully used as a theorem prover, in which case, it is highly probable that the process will never terminate. The proof evidence supports, though, anonymous variables anywhere in the decide tree. One can also completely omit the decide tree, in which case the certifier reconstruct it while certifying the proof, or decide to supply any amount of information, which

[2]Clearly, if one considers the two-sided sequent version of $LS$ 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 $LS$ sequents into one-sided sequents, as shown, e.g., in [6].

[3]/src/fpc/modal/tableaux.mod.

will be used in order to execute a more adequate proof checking process.

## 3.3 Implementation and examples

In the previous section, we have presented key features of `checkers` and demonstrated them on an example. In this section, we will discuss the implementation in more details and explain how `checkers` can be used or extended.

`checkers` is implemented in λProlog and takes advantage of several of its features. First, logic programming allows us to denote both kernel, proof evidence and FPC specifications in the same language. This increases trust as it allows us to execute the code without any additional translation. At the same time, the kernel can be implemented in any programming language. In addition, we can take advantage of having variables as part of the syntax in order to abstract over arbitrary information in the proof evidence. Second, λProlog, being based on logical predicates, is highly suitable for the definition of the trusted kernel. Unlike other programming languages, notions such as substitution, unification and proof search are defined on the language level and need not be implemented. This enables us both to have a faithful translation of logical calculi into code as well as the ability to separate proof search from calculus definitions. Technically, we support two different implementations of λProlog, Teyjus[4] and ELPI[5] and therefore `checkers` enjoys a higher-level of trust since only the calculus definitions should be trusted and not the proof search implementation. In practice, since `checkers` is using a modular design which exploits several weaknesses of Teyjus, we recommend to use ELPI[6] for executing `checkers`. Due to some differences in their syntax and logical structure (ELPI uses full type inference while Teyjus uses type checking), we provide two different entry points. When using Teyjus, `checkers` can be executed as follows:

```
./prover-teyjus.sh <cert>
```

While the ELPI interpreter is called using:

```
./prover-elpi.sh <cert> [--debug]
```

[4] `http://teyjus.cs.umn.edu/`

[5] `http://lpcic.gforge.inria.fr/`

[6] The ELPI version used can be downloaded from here `http://lpcic.gforge.inria.fr/elpi-LFMTP16.tar.gz`

The input to the program is the name of a λProlog module which is already included in the program path. For example, in order to certify the proof evidence named $modlab - full - s5$ in the file `modlab-full-s5.mod`, one need to execute:

```
./prover-elpi.sh modlab-full-s5
```

The optional debug flag tells the interpreter to output the whole proof search executed. Using this flag, one can easily see the difference between the deterministic application of rules on the fully verbose proof evidences versus the required proof search employed in the minimal versions.

`checkers` is distributed with a set of 20 examples which are stored in the `src/test/modal` folder. This folder is already included in the program path so examples can be immediately executed. The examples are divided into tableau (modtab) and labeled (modlab) examples as well as into fully verbose (full) examples and those containing only the essential map (min).

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[7], a free variable modal tableau prover written in Prolog.

A typical proof evidence file can be seen in Fig. 13. The evidence starts with the name of the module and then specify the kernel and then FPC specifications required in order to certify it. Enhancing `checkers` with new kernels as well as new FPC specification does not, therefore, require changing the current code. The body of the evidence then contains the textual name, the list of axioms used (indexed by a name), the decision tree and the essential map. The last component is the state (whose store list is initialized with the indices to use in order to store the axioms).

Since the example portraits a minimal proof evidence, with no information about the tree of rules to be applied, we simply denote the decide tree with `dectree eind _`. We explicitly name the index of the root node (for easier debugging) but leave the rest of the tree implicit (using the Prolog anonymous variable notation `_`). The remaining part of the evidence consists of the essential map, which includes the maps for □-◇ applications,

[7] `http://formal.iti.kit.edu/beckert/modlean/`

closure applications and axiom instantiations.

A more complex example is given in Figure 14. The example describes a simple situation, in which a condition of fairness, usually analyzed in the context of temporal logics, is verified in the logic $S4$. The formula proved is the following:

$$\Box(req \supset \Diamond ena) \supset (\Box\Diamond req \supset \Box\Diamond ena)$$

Intuitively, it expresses the fact that if we are in a state such that whenever a process makes a *request* then it is eventually *enabled*, then the condition of fairness is verified. The notion of fairness used here says that if a request is made infinitely often, then the process is enabled infinitely often. In Figure 14, we present a labeled derivation of the theorem (in an equivalent formulation in negation normal form) in the system $LS_{S4}$. The corresponding full proof evidence is given in Figure 15.

The version of `checkers` used in the above examples can be obtained from the `gandalf2017` branch on its Github page[8].

# 4 Concluding remarks and future work

In this paper, we have presented an approach for certifying labeled modal proofs, given either in the form of labeled sequents or in the form of prefixed tableaux. In both cases, we have considered a range of well-known modal logics, going from the basic logic $K$ to all those logics characterized by frames that can be defined by combinations of the properties of reflexivity, symmetry and transitivity, including in particular the logics $S4$ and $S5$. Moreover, we have shown theoretically how the approach can be applied in general to all the modal logics whose frames are defined by geometric properties. This might require some small adjustments to the framework, according to the nature of the axiom and of the corresponding sequent/tableau rules. E.g., in the case of axioms/rules that allow for generating new worlds along a proof, like seriality, which says that for each world there exists a reachable one, a slight modification of the way labels are represented is required, since labels are now introduced not necessarily by the application of $\Box$-rules.

Our framework could also be quite easily extended to consider modal logics with more than one accessibility relation, like those used for reasoning on multi-agent systems, by simply introducing in our translation one first-order

[8] `https://github.com/proofcert/checkers/tree/gandalf2017`

binary predicate for each such an accessibility relation and by parameterizing the application of modal rules with respect to a specific predicate.

In the spirit of generality pursued by the ProofCert project, we plan to consider in the future proof systems for modal logics based on non-labeled formalisms. In [22], a general focused framework for emulating modal proof systems based on different formalisms, like sequent calculi and nested sequent calculi, has been introduced. We believe that our approach, being based on a focused sequent calculus as kernel, can be extended/adapted to implement such a framework as well as to deal with further formalisms, like resolution. Since many modal resolution provers are based on a translation into a first-order language, in fact we expect to be able to reuse part of this work also in that setting.

Technically speaking, supporting other proof evidences and calculi amounts to writing new FPC specifications. The modular structure of `checkers` allows for having flexible relationships between the different FPC specifications, e.g., by composing and extending them, ideally by using a layered architecture. Given the expressivity of labeled calculi in the context of modal logics, and the numerous results of encoding of other proof systems into such calculi, we believe that the FPC designed here can provide a basis on top of which to build a modular architecture able to capture further proof formats and formalisms.

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

# References

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

[2] H. J. Ohlbach, A resolution calculus for modal logics, in: E. L. Lusk, R. A. Overbeek (Eds.), 9th International Conference on Automated Deduction, Argonne, Illinois, USA, May 23-26, 1988, Proceedings, Vol. 310 of Lecture Notes in Computer Science, Springer, 1988, pp. 500–516. doi:10.1007/BFb0012852.

[3] D. Miller, Proofcert: Broad spectrum proof certificates, an ERC Advanced Grant funded for the five years 2012-2016 (Feb. 2011).

[4] Z. Chihani, T. Libal, G. Reis, The proof certifier checkers, in: H. de Nivelle (Ed.), Automated Reasoning with Analytic Tableaux and Related Methods - 24th International Conference, TABLEAUX 2015, Wrocław, Poland, September 21-24, 2015. Proceedings, Vol. 9323 of Lecture Notes in Computer Science, Springer, 2015, pp. 201–210. doi:10.1007/978-3-319-24312-2_14.

[5] C. Liang, D. Miller, Focusing and polarization in linear, intuitionistic, and classical logics, Theor. Comput. Sci. 410 (46) (2009) 4747–4768. doi:10.1016/j.tcs.2009.07.041.

[6] D. Miller, M. Volpe, Focused labeled proof systems for modal logic, in: M. Davis, A. Fehnker, A. McIver, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings, Vol. 9450 of Lecture Notes in Computer Science, Springer, 2015, pp. 266–280. doi:10.1007/978-3-662-48899-7_19.

[7] S. Negri, Proof analysis in modal logic, J. Philosophical Logic 34 (5-6) (2005) 507–544. doi:10.1007/s10992-005-2267-3.

[8] T. Libal, M. 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., 2016, pp. 257–271.

[9] M. Boespflug, Q. Carbonneaux, O. Hermant, The $\lambda\pi$-calculus modulo as a universal proof language, in: the Second International Workshop on Proof Exchange for Theorem Proving (PxTP 2012), Vol. 878, 2012, pp. pp–28.

[10] R. Cauderlier, P. Halmagrand, Checking zenon modulo proofs in dedukti, in: C. Kaliszyk, A. Paskevich (Eds.), Proceedings Fourth Workshop on Proof eXchange for Theorem Proving, PxTP 2015, Berlin, Germany, August 2-3, 2015., Vol. 186 of EPTCS, 2015, pp. 57–73. doi:10.4204/EPTCS.186.7.

[11] D. Miller, G. Nadathur, Programming with Higher-Order Logic, Cambridge University Press, 2012.

[12] J. Andreoli, Logic programming with focusing proofs in linear logic, J. Log. Comput. 2 (3) (1992) 297–347. doi:10.1093/logcom/2.3.297.

[13] H. Sahlqvist, Completeness and correspondence in first and second order semantics for modal logic, in: N. H. S. Kanger (Ed.), Proceedings of the Third Scandinavian Logic Symposium, 1975, pp. 110–143.

[14] P. Blackburn, J. Van Benthem, Modal logic: a Semantic Perspective, in: F. W. Patrick Blackburn, Johan van Benthem (Ed.), Handbook of Modal Logic, Elsevier, 2007, pp. 1–82.

[15] D. M. Gabbay, Labelled Deductive Systems, Clarendon Press, 1996.

[16] M. Fitting, Modal proof theory, in: P. Blackburn, J. van Benthem, F. Wolter (Eds.), Handbook of Modal Logic, Elsevier, 2007, pp. 85–138.

[17] J. A. Robinson, A machine-oriented logic based on the resolution principle, J. ACM 12 (1) (1965) 23–41. doi:10.1145/321250.321253.

[18] M. Baaz, A. Leitsch, Complexity of resolution proofs and function introduction, Ann. Pure Appl. Logic 57 (3) (1992) 181–215. doi:10.1016/0168-0072(92)90042-X.

[19] S. Reeves, Semantic tableaux as a framework for automated theorem-proving, in: On Advances in Artificial Intelligence, John Wiley & Sons, Inc., New York, NY, USA, 1987, pp. 125–139.

[20] B. Beckert, R. Goré, Free-variable tableaux for propositional modal logics, Studia Logica 69 (1) (2001) 59–96. doi:10.1023/A:1013886427723.

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

[22] S. Marin, D. Miller, M. 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.

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

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

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

IDENTITY RULES

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

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 $\Xi, \Theta, \Gamma, B$. The symbol $P_a$ denotes a positive atomic formula.

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

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

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

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

Figure 2: $LS_K$: a labeled sequent system for the modal logic $K$

$$\frac{xRx, \Gamma \vdash \Delta}{\Gamma \vdash \Delta}\ T_{LS} \qquad \frac{xRz, xRy, yRz, \Gamma \vdash \Delta}{xRy, yRz, \Gamma\ \vdash \Delta}\ 4_{LS} \qquad \frac{yRx, xRy, \Gamma \vdash \Delta}{xRy, \Gamma \vdash \Delta}\ B_{LS}$$

Figure 3: Rules for capturing relational properties.

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{}{xRy, xRz, zRx, zRy \vdash y : \neg p, z : \Diamond p, y : p}\ init}{xRy, xRz, zRx, zRy \vdash y : \neg p, z : \Diamond p}\ R\Diamond}{xRy, xRz, zRx \vdash y : \neg p, z : \Diamond p}\ 4_{LS}}{xRy, xRz \vdash y : \neg p, z : \Diamond p}\ B_{LS}}{xRy \vdash y : \neg p, x : \Box\Diamond p}\ R\Box}{\vdash x : \Box\neg p, x : \Box\Diamond p}\ R\Box}{\vdash x : \Box\neg p \vee \Box\Diamond p}\ R\vee$$

Figure 4: Derivation of the axiom for euclideaness in $LS_{S5}$.

Classical rules

$$\frac{\sigma : A \wedge B}{\sigma : A,\ \sigma : B}\ \wedge_{PT} \qquad \frac{\sigma : A \vee B}{\sigma : A \quad | \quad \sigma : B}\ \vee_{PT}$$

Modal rules

$$\frac{\sigma : \Box A}{\sigma.n : A}\ \Box_{PT} \qquad \frac{\sigma : \Diamond A}{\sigma.n : A}\ \Diamond_{PT}$$

In $\Box_{PT}$, $\sigma.n$ is used. In $\Diamond_{PT}$, $\sigma.n$ is new.

Figure 5: $PT_K$: a prefixed tableau system for the modal logic $K$

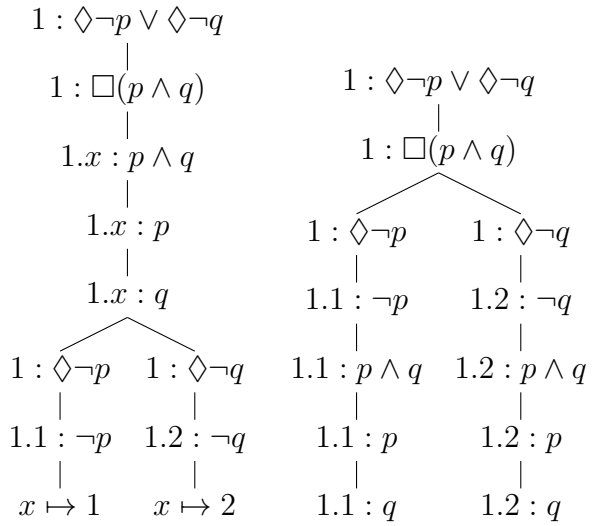
$$\frac{\sigma : \Box A}{\sigma : A}\ T_{PT} \qquad \frac{\sigma.n : \Box A}{\sigma : A}\ B_{PT} \qquad \frac{\sigma : \Box A}{\sigma.n : \Box A}\ 4_{PT} \qquad \frac{\sigma.n : \Box A}{\sigma : \Box A}\ 4r_{PT}$$

In $\Box_{PT}$, $\sigma.n$ is used. In $\Diamond_{PT}$, $\sigma.n$ is new.

Figure 6: Prefixed tableau rules for modal logics extending $K$

$$\begin{array}{c} 1 : \Diamond p \wedge \Diamond\Box\neg p \\ | \\ 1 : \Diamond p \\ | \\ 1 : \Diamond\Box\neg p \\ | \\ 1.1 : p \\ | \\ 1.2 : \Box\neg p \\ | \\ 1 : \Box\neg p \\ | \\ 1.1 : \neg p \end{array}$$

Figure 7: Prefixed tableau derivation for the axiom of euclideaness




(a) Example of FV tableau

(b) Example of prefixed tableau

Figure 8: Prefixed and free-variable prefixed tableau derivations

```
eind : index                              relind : index
lind : index -> index                     rind   : index -> index
bind : index -> index -> index

rel : term -> term -> atm

dectree : dectree index -> list dectree -> dectree.

% a map between dia indices and box indices
diabox-entry index -> index -> diabox-entry.
diabox-map list diabox-entry -> diabox-map.

% a map between init indices and a complementary index
init-entry index -> index -> init-entry.
init-map list init-entry -> init-map.

% a map between axioms and the indices relating to their instantiations
axiom-entry index -> list index -> axiom-entry.
axiom-map list axiom-entry -> axiom-map.

modlab-cert dectree -> diabox-map -> init-map -> axiom-map -> cert.
```

Figure 9: Type declaration for the LS FPC specification.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\texttt{lind(lind(rind(eind)))}}}{\texttt{lind(rind(eind))}}}{4_{LS}}}{B_{LS}}}{\texttt{rind(eind)}}}{\texttt{lind(eind)}}}{\texttt{eind}}$$

Figure 10: The decide tree component of the certificate for the proof in Fig. 4.

```
(modlab-cert
  (dectree eind [dectree (lind eind) [dectree (rind eind) [dectree symm-ind
    [dectree trans-ind [dectree (lind (rind eind)) [dectree (bind (lind (
    rind eind)) (lind eind)) []]]]]]])
  (diabox-map [diabox-entry (lind (rind eind)) (lind eind)])
  (init-map [init-entry (bind (lind (rind eind)) (lind eind)) (lind (lind
    eind))]) (axiom-map [axiom-entry symm-ind [default-ind, rind eind],
    axiom-entry trans-ind [rind eind, default-ind, lind eind]]))
```

Figure 11: The full certificate for the proof in Fig. 4.

```
decide_ke (modlab-cert (dectree I D) M1 M2 M5 Bnd (state _ M3 M4)) I (
    modlab-cert (dectree I D) M1 M2 M5 Bnd (state [] M3 M4')).
store_kc (modlab-cert DT M1 M2 M5 Bnd (state [H|T] M3 M4)) _ H (modlab-cert
    DT M1 M2 M5 Bnd (state T M3 [decide-bound-entry H Bnd | M4])).
release_ke C C.
initial_ke (modlab-cert (dectree relind []) _ _ _ _ _) relind.
initial_ke (modlab-cert (dectree I _) _ (init-map M2) _ _ _) O :- member (
    init-entry I O) M2.
orNeg_kc (modlab-cert (dectree I [D]) M1 M2 M5 Bnd (state [] M3 M4)) _ (
    modlab-cert D M1 M2 M5 Bnd (state [lind I, rind I] M3 M4)).
orNeg_kc (modlab-cert D M1 M2 M5 Bnd (state [S] M3 M4)) ((n (rel _ _) !-! _)
    ) (modlab-cert D M1 M2 M5 Bnd (state [relind,S] M3 M4)).
orNeg_kc (modlab-cert D M1 M2 M5 Bnd (state [S] M3 M4)) ((p (rel _ _) !-! _)
    ) (modlab-cert D M1 M2 M5 Bnd (state [relind,S] M3 M4)).
orNeg_kc (modlab-cert D M1 M2 M5 Bnd (state S M3 M4)) _ (modlab-cert D M1 M2
     M5 Bnd (state S M3 M4)).
andPos_k (modlab-cert (dectree I D) M1 M2 M5 Bnd S) (_ &+& (p (rel _ _)))
    left-first
  (modlab-cert (dectree relind []) _ _ _ _ (state [] _ _)) (modlab-cert (
    dectree relind []) M1 M2 M5 Bnd S).
andPos_k (modlab-cert (dectree I D) M1 M2 M5 Bnd S) _ left-first
  (modlab-cert (dectree relind []) M1 M2 M5 Bnd (state [] _ _)) (modlab-cert
     (dectree I D) M1 M2 M5 Bnd S).
all_kc (modlab-cert (dectree I [S]) M1 M2 M5 Bnd (state [] M3 M4))
  (Eigen\ modlab-cert S M1 M2 M5 Bnd (state [lind I] [eigen-entry I Eigen|M3
    ] M4)) :-
  member (eigen-entry I _) M3.
all_kc (modlab-cert (dectree I [S]) M1 M2 M5 Bnd (state [] M3 M4))
  (Eigen\ modlab-cert S M1 M2 M5 Bnd (state [lind I] [eigen-entry I Eigen|M3
    ] M4)).
some_ke (modlab-cert (dectree I [S]) (diabox-map M1) M2 M5 Bnd (state [] M3
    M4)) Eigen
  (modlab-cert S (diabox-map M1) M2 M5 Bnd (state [bind I O] M3 M4)) :-
  (member (diabox-entry I O) M1, member (eigen-entry O Eigen)  M3).
some_ke (modlab-cert (dectree I [S]) M1 M2 (axiom-map M5) Bnd (state _ M3 M4
    )) Eigen
  (modlab-cert S M1 M2 (axiom-map M5') Bnd (state [relind] M3 M4)) :-
  (memb_and_rest (axiom-entry I [O]) M5 M5', member (eigen-entry O Eigen)
    M3).
some_ke (modlab-cert (dectree I [S]) M1 M2 (axiom-map M5) Bnd (state _ M3 M4
    )) Eigen
  (modlab-cert (dectree I [S]) M1 M2 (axiom-map [axiom-entry I [Q|Ls] | M5
    ']) Bnd (state [relind] M3 M4)) :-
  (memb_and_rest (axiom-entry I [O,Q|Ls]) M5 M5', member (eigen-entry O
    Eigen)  M3).
```

Figure 12: Definition of the LS FPC specification.

```
module modtab-min-s5.
accumulate tableaux.
accumulate lkf-kernel.
modalProblem "Tableau␣Problem␣S5␣-␣minimal␣proof␣evidence"
[(pr symm-ind (some (x\ some (y\ (p (rel x y) &+& n (rel y x)) ) ))), (pr
    trans-ind (some (x\ some (y\ some z\ ((p (rel x y) &+& p (rel y z)) &+&
    n (rel x z) ) ))))]
(box (-- p1) !! box (dia (++ p1)))
(modtab-cert
  (dectree eind _)
  (diabox-map [diabox-entry (lind (rind eind)) (lind eind)])
  (init-map [init-entry (bind (lind (rind eind)) (lind eind)) (lind (lind
    eind))])
(axiom-map [axiom-entry symm-ind [default-ind, rind eind], axiom-entry
    trans-ind [rind eind, default-ind, lind eind]])
 (snum (snum znum))
 (state [trans-ind, symm-ind] [eigen-entry default-ind zero] []) ).
```

Figure 13: A minimal tableau proof evidence of the proof in Fig. 7.

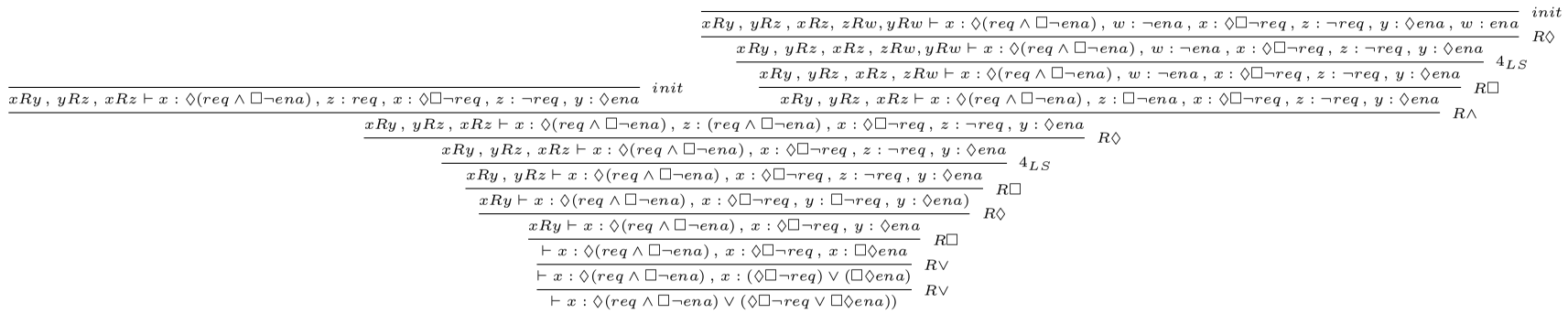



Figure 14: Fairness-inspired example of derivation in $LS_{S4}$.

```
module modlab-full-s4.
accumulate labeled.
accumulate lkf-kernel.
modalProblem "Modal␣problem␣S4␣full␣proof␣evidence"
[(pr trans-ind (some (x\ some (y\ some z\ ((p (rel x y) &+& p (rel y z)) &+&
     n (rel x z) ) )))))]
((dia (++ req && box (-- ena))) !! (dia (box (-- req)) !! box (dia (++ ena))
    ) )
(modlab-cert
  (dectree eind [dectree (rind eind) [dectree (rind (rind eind)) [dectree (
    lind (rind eind)) [dectree (bind (lind (rind eind)) (rind (rind eind)))
    [dectree trans-ind [dectree (lind eind) [dectree (bind (lind eind) (bind
     (lind (rind eind)) (rind (rind eind)))) [dectree (lind (bind (lind eind
    ) (bind (lind (rind eind)) (rind (rind eind))))))[], dectree (rind (bind
    (lind eind) (bind (lind (rind eind)) (rind (rind eind))))) [dectree
    trans-ind [dectree (lind (rind (rind eind))) [dectree (bind (lind (rind
    (rind eind))) (rind (bind (lind eind) (bind (lind (rind eind)) (rind (
    rind eind)))))) [] ]]]]]]]]]]]])
  (diabox-map [diabox-entry (lind (rind eind)) (rind (rind eind)),
    diabox-entry (lind eind) (bind (lind (rind eind)) (rind (rind eind))),
    diabox-entry (lind (rind (rind eind))) (rind (bind (lind eind) (bind (
    lind (rind eind)) (rind (rind eind))))))])
  (init-map [init-entry (lind (bind (lind eind) (bind (lind (rind eind)) (
    rind (rind eind))))) (lind (bind (lind (rind eind)) (rind (rind eind))))
    , init-entry (bind (lind (rind (rind eind))) (rind (bind (lind eind) (
    bind (lind (rind eind)) (rind (rind eind)))))) (lind (rind (bind (lind
    eind) (bind (lind (rind eind)) (rind (rind eind)))))])
  (axiom-map [axiom-entry trans-ind [default-ind, rind (rind eind), bind (
    lind (rind eind)) (rind (rind eind))], axiom-entry trans-ind [rind (rind
     eind), bind (lind (rind eind)) (rind (rind eind)), rind (bind (lind
    eind) (bind (lind (rind eind)) (rind (rind eind))))]])
 (snum (snum znum))
 (state [trans-ind] [eigen-entry default-ind zero] []) ).
```

Figure 15: A full labeled sequent proof evidence of the proof in Fig. 14.