Exploring Properties of Normal Multimodal Logics in Simple Type Theory with LEO-II[1]

CHRISTOPH BENZMÜLLER AND LAWRENCE C. PAULSON

To Peter B. Andrews

1 Introduction

There are two well investigated approaches to automate reasoning in modal logics: the direct approach and the translational approach. The direct approach [6, 7, 14, 27] develops specific calculi and tools for the task; the translational approach [29, 30] transforms modal logic formulas into first-order logic and applies standard first-order tools. Embeddings of modal logics into higher-order logic, however, have not yet been widely studied, although multimodal logic can be regarded as a natural fragment of simple type theory. Gallin [15] appears to mention the idea first. He presents an embedding of modal logic into a 2-sorted type theory. This idea is picked up by Gamut [16] and a related embedding has recently been studied by Hardt and Smolka [17]. Carpenter [12] proposes to use lifted connectives, an idea that is also underlying the embeddings presented by Merz [26], Brown [11], Harrison [18, Chap. 20], and Kaminski and Smolka [22].

In this paper we pick up and extend the embedding of multimodal logics in simple type theory as studied by Brown [11]. The starting point is a characterization of multimodal logic formulas as particular λ-terms in simple type theory. A distinctive characteristic of the encoding is that the definiens of the $\Box_R$ operator λ-abstracts over the accessibility relation R. We illustrate that this supports the formulation of meta properties of encoded multimodal logics such as the correspondence between certain axioms and properties of the accessibility relation R. We show that some of these meta properties can even be efficiently automated within our higher-order theorem prover LEO-II [9] via cooperation with the first-order automated

[1]This work was supported by EPSRC grant EP/D070511/1 (LEO II: An Effective Higher-Order Theorem Prover) at Cambridge University, UK.

theorem prover E [32]. We also discuss some challenges to higher-order reasoning implied by this application direction. Moreover, we extend the presented embedding to first-order and higher-order quantified multimodal logics.

2 Simple Type Theory

Classical higher-order logic or simple type theory [4, 13] is a formalism built on top of the simply typed λ-calculus. The set $\mathcal{T}$ of simple types is usually freely generated from a set of basic types $\mathcal{BT} = \{o, \iota\}$ using the function type constructor $\rightarrow$. Here we allow an arbitrary but fixed number of additional base types to be specified.

For formulae, we start with a set of typed variables $X_\alpha, Y, Z, X^1_\beta, X^2_\gamma \ldots$ and a set of typed constants $c_\alpha, f_{\alpha\rightarrow\beta}, \ldots$. The set of constants includes the primitive logical connectives $\neg_{o\rightarrow o}$, $\vee_{o\rightarrow o\rightarrow o}$ and $\Pi_{(\alpha\rightarrow o)\rightarrow o}$ (abbreviated Π^α) and $=_{\alpha\rightarrow\alpha\rightarrow o}$ (abbreviated $=^\alpha$) for each type α.[1] Other logical connectives can be defined in terms of the these primitive ones in the usual way. In the remainder of this paper, we assume that any signature Σ we consider at least contains the primitive logical connectives $\neg_{o\rightarrow o}$, $\vee_{o\rightarrow o\rightarrow o}$ and $\Pi_{(\alpha\rightarrow o)\rightarrow o}$.

Terms and formulae are constructed from typed variables and constants using application and λ-abstraction. We use Church's dot notation so that $\centerdot$ stands for an implicit left bracket whose mate is as far to the right as possible (consistent with explicit brackets). We use infix notation $A \vee B$ for $((\vee A)B)$ and binder notation $\forall X_\alpha \centerdot A$ for $(\Pi^\alpha(\lambda X_\alpha \centerdot A_o))$.

Standard and Henkin semantics of simple type theory is well understood and thoroughly documented in the literature [1, 2, 8, 19].

One particular challenge for the complete automation of higher-order logic is *primitive substitution* [3] or splitting [20]. Primitive substitutions blindly guess some logical structure for free predicate or set variables in a clause that cannot be synthesized otherwise, and they introduce new free variables in order to delay some further decisions. The instantiation of the new and the remaining free variables is ideally supported by higher-order pre-unification. Generally, however, the primitive substitution process has to be iterated which leads to very challenging search space for clause sets containing many free variables. Some of the proof problems discussed in this paper require simple primitive substitutions.

[1] Note that there are infinitely many different Π^α and $=^\alpha$ introduced here .

3 Encoding Multimodal Logics in Simple Type Theory

Simple type theory is an expressive logic and it is thus no surprise that modal logic can be encoded in several ways in it. Harrison [18], for instance, presents a 'deep embedding' of modal logics by formalizing standard Kripke semantics and a 'shallow embedding' of the temporal logic LTL. The latter encoding more naturally exploits the expressiveness of higher-order logic. Harrison's shallow embedding is an instance of the encoding due to Brown [11]. Here we adapt and further extend Brown's suggestion and show that this approach is well suited for reasoning within and about modal logics.

The idea of the encoding is simple: Choose a base type — we choose ι — to denote the set of all possible worlds. Certain formulas of type $\iota \to o$ then correspond to multimodal logic expressions. The modal operators $\neg$, $\vee$, and $\Box_R$ become λ-terms of types $(\iota \to o) \to (\iota \to o)$, $(\iota \to o) \to (\iota \to o) \to (\iota \to o)$, and $(\iota \to \iota \to o) \to (\iota \to o) \to (\iota \to o)$ respectively. Note that $\neg$ forms the complement of a set of worlds, while $\vee$ forms the union of two such sets.

DEFINITION 1 (Propositional Multimodal Logic Λ_0^{mm}). Given a signature Σ, we define the the set Λ_0^{mm} of propositional multimodal logic propositions as follows.

1. We introduce the logical constants of Λ_0^{mm} as abbreviations for the following λ-terms:
$$\begin{aligned} \neg_{(\iota\to o)\to(\iota\to o)} &= \lambda A_{\iota\to o}.\lambda X_\iota.\neg A\,X \\ \vee_{(\iota\to o)\to(\iota\to o)\to(\iota\to o)} &= \lambda A_{\iota\to o}.\lambda B_{\iota\to o}.\lambda X_\iota.A\,X \vee B\,X \\ \Box_{R\ (\iota\to\iota\to o)\to(\iota\to o)\to(\iota\to o)} &= \\ & \lambda R_{\iota\to\iota\to o}.\lambda A_{\iota\to o}.\lambda X_\iota.\forall Y_\iota.R\,X\,Y \Rightarrow A\,Y \end{aligned}$$
2. We define the set of Λ_0^{mm}-propositions as the smallest set of simply typed λ-terms for which the following hold:
   - Each constant $p_{\iota\to o} \in \Sigma$ is an atomic Λ_0^{mm}-proposition.
   - If ϕ and ψ are Λ_0^{mm}-propositions, then so are $\neg\,\phi$, $\phi \vee \psi$ and $\Box_R\,\phi$, where $\neg$, $\vee$, and $\Box_R$ are defined as above and where R is a term of type $\iota \to \iota \to o$.
3. The propositional multimodal logic operators $\Rightarrow$, $\Leftrightarrow$, $\Diamond_r$, etc. can be defined in terms of $\neg$, $\vee$ and $\Box_R$ in the usual way.

Note that the encoding of the modal operator $\Box_R$ depends explicitly on an accessibility relation R of type $\iota \to \iota \to o$ given as its first argument. Hence, we basically introduce a generic framework for modeling multimodal

logics. This idea is where Brown [11] differs from the LTL encoding of Harrison. The latter chooses the interpreted type *num* of numerals and then uses the predefined relation $\leq$ over numerals as a fixed accessibility relation in the definitions of $\Box$ and $\Diamond$.

By making the dependency of $\Box_R$ and $\Diamond_R$ on the accessibility relation R explicit, we can formalize and automatically prove some properties of multimodal logics in simple type theory, as we will later illustrate.

Given a signature Σ containing some constants $a_{\iota\to o}, b_{\iota\to o}$ (basic modal propositions) and $r_{\iota\to\iota\to o}, s_{\iota\to\iota\to o}$ (accessibility relation constants) we can formulate statements in Λ_0^{mm}, such as these:

$$\Box_r\ \top \qquad \Box_r\, a \Rightarrow \Box_r\, a \qquad \Box_r\, a \Rightarrow \Box_s\, a \qquad \Box_s\,(\Box_r\, a \Rightarrow \Box_r\, a)$$

We assume that $\Box_r$ binds more strongly than the propositional connectives and, hence, $\Box_r\, a \Rightarrow \Box_r\, a$ stands for $(\Box_r\, a) \Rightarrow (\Box_r\, a)$.

Next, we define validity of modal logic expressions $A_{\iota\to o} \in \Lambda_0^{mm}$: formula A is valid iff for all possible worlds W_ι we have $W \in A$, that is, iff $A\,W$ holds.

DEFINITION 2 (Validity, Satisfiability, Falsifiable). We define the following notions as simply typed λ-terms:

$$\begin{aligned}
\text{valid} &:= \lambda A_{\iota\to o}.\forall W_\iota.A\,W\\
\text{satisfiable} &:= \lambda A_{\iota\to o}.\exists W_\iota.A\,W\\
\text{falsifiable} &:= \lambda A_{\iota\to o}.\exists W_\iota.\neg A\,W\\
\text{unsatisfiable} &:= \lambda A_{\iota\to o}.\forall W_\iota.\neg A\,W
\end{aligned}$$

4 Solving Simple Λ_0^{mm}-problems with LEO-II

We can now exploit this framework to automatically analyze modal logic formulas within a higher-order theorem prover such as our LEO-II. Table 1 presents performance results for LEO-II on some example problems for the multimodal logic K. Since LEO-II cooperates with a first-order theorem prover, we actually present two times in the table. The first one is the total reasoning time used by both cooperating systems. The second time is that used by LEO-II alone. (Hence their difference is the time spent in the first-order theorem prover.) The times are given in seconds. All experiments reported in this paper were conducted with version v0.9 of LEO-II on a notebook computer with a Intel Pentium 1.60GHz processor with 1GB memory running Linux.

K^c cannot be solved by LEO-II, which is fortunate: this statement is clearly not valid without imposing restrictions on r and s. The other state-

Table 1. Runtimes for Proving Modal Logic Theorems

	problem	LEO-II+E / LEO-II
K^a	valid($\Box_r\ \top$)	0.024/0.008
K^b	valid($\Box_r\, a \Rightarrow \Box_r\, a$)	0.032/0.012
K^c	valid($\Box_r\, a \Rightarrow \Box_s\, a$)	–
K^d	valid($\Box_s\, (\Box_r\, a \Rightarrow \Box_r\, a)$)	0.033/0.016
K^e	valid($\Box_r\, (a \wedge b) \Leftrightarrow (\Box_r\, a \wedge \Box_r\, b)$)	0.051/0.020
K^f	valid($\Diamond_r\, (a \Rightarrow b) \Rightarrow \Box_r\, a \Rightarrow \Diamond_r\, b$)	0.038/0.016
K^g	valid($\neg \Diamond_r\, a \Rightarrow \Box_r\, (a \Rightarrow b)$)	0.031/0.012
K^h	valid($\Box_r\, b \Rightarrow \Box_r\, (a \Rightarrow b)$)	0.032/0.012
K^i	valid($(\Diamond_r\, a \Rightarrow \Box_r\, b) \Rightarrow \Box_r\, (a \Rightarrow b)$	0.032/0.012
K^j	valid($(\Diamond_r\, a \Rightarrow \Box_r\, b) \Rightarrow (\Box_r\, a \Rightarrow \Box_r\, b)$)	0.035/0.016
K^k	valid($(\Diamond_r\, a \Rightarrow \Box_r\, b) \Rightarrow (\Diamond_r\, a \Rightarrow \Diamond_r\, b)$)	0.035/0.016

ments expand into trivially refutable problems and they are quickly solved by the LEO-II+E cooperation.

We explain how the cooperation between LEO-II and E solves problem K^d. When the problem is initially input to LEO-II, the prover creates the following clause:

$$\mathcal{C}_1 : \quad [\text{valid}(\Box_s\, (\Box_r\, a \Rightarrow \Box_r\, a))]^F$$

This clause consists of one negated literal, as indicated by the superscript F. The atom of the literal, enclosed in square brackets, consists of the original problem statement. LEO-II unfolds the definitions and thereby rewrites this clause into

$$\begin{aligned}\mathcal{C}_2 : \quad & [\forall X^0_\iota\, .\, \forall X^1_\iota\, .\, : \neg (s\, X^0\, X^1) \\ & \vee ((\neg (\forall X^2_\iota\, .\, \neg (r\, X^1\, X^2) \vee (a\, X^2))) \vee (\forall X^2_\iota\, .\, \neg (r\, X^1\, X^2) \vee (a\, X^2)))]^F .\end{aligned}$$

This clause is obviously not in normal form yet. Clause normalization is the next step performed in LEO-II. This produces the following four normal form clauses, where the sk^j are Skolem constants and the V^1 is a variable:

$$\begin{aligned}\mathcal{C}_3 : \quad & [s\, sk^1\, sk^2]^T \\ \mathcal{C}_4 : \quad & [a\, sk^3]^F \\ \mathcal{C}_5 : \quad & [r\, sk^2\, sk^3]^T \\ \mathcal{C}_6 : \quad & [a\, V^1]^T \vee [r\, sk^2\, V^1]^F\end{aligned}$$

Subsequent to these pre-processing steps, LEO-II calls the first-order theorem prover E. The translation from higher-order to first-order logic is based on the ideas of Kerber [23].[2] This translation introduces a family of application operators @ and encodes type information into their names. For example, the operator $@_{(\beta\alpha)\_\beta}$ encodes the information that its first argument has function type $\beta \to \alpha$ and its second argument type β. Each clause in the domain of the translation is passed after executing the transformation to the prover E. In our example, four first-order clauses are generated and passed to E:

$$\begin{aligned}
\mathcal{C}_{3'}: &\quad @_{(io)_i}(@_{(i(io))_i}(s, sk^1), sk^2)\\
\mathcal{C}_{4'}: &\quad @_{(\iota o)_\iota}(a, sk^3)\\
\mathcal{C}_{5'}: &\quad @_{(io)_i}(@_{(i(io))_i}(r, sk^2), sk^3)\\
\mathcal{C}_{6'}: &\quad @_{(\iota o)_\iota}(a, V^1) \vee (r, sk^2), V^1)
\end{aligned}$$

E immediately finds a refutation for this trivial set of clauses and signals success back to LEO-II. This in turn triggers LEO-II to stop its proof search and to report success to the user. The proof protocol generated by LEO-II for this problem is given in Appendix A.

LEO-II alone, not cooperating with E, would also quickly refute this trivial set of clauses, but here E signals success before LEO-II even starts its own main proof loop. Generally the subset of clauses in LEO-II's search space that can be passed to E usually becomes much bigger and these are the cases where the cooperation between LEO-II and the first-order prover really pays off: the first-order specialist prover may quickly find the refutation while LEO-II alone gets stuck in its much more challenging search space. In the refutation proof of problem K^e, for example, LEO-II passes 24 first-order clauses to E, which then generates 322 further clauses before signaling that a refutation has been found. While the search space here is still trivial for both, LEO-II and E, we will see examples in Section 5 where E generates more than 20000 clauses before it finds a refutation for the subproblem passed to it by LEO-II.

More information of the design of LEO-II is provided in Benzmüller et al. [9]. In general terms, the first-order specialist provers support LEO-II by periodically trying to detect subsets of clauses in LEO-II's search space that can be quickly refuted by them.

This simple idea can be further generalized

(i) by realizing analogous cooperations with specialist reasoners working for other interesting and efficiently mechanizable fragments of higher-

[2] Further translations have been discussed in the literature; see for example [21, 24, 25].

order logic such as propositional logic, guarded fragment, monadic second-order logic, etc.,

(ii) by allowing LEO-II and these reasoners to run in parallel, and

(iii) by supporting alternative translations to first-order logic and other fragments of higher-order logic.

In fact, LEO-II already provides a link to the first-order theorem prover SPASS as an alternative to E. And it also already supports an alternative translation to first-order logic. In order to achieve the distribution goals we plan to adapt the OANTS system [10] to LEO-II.

We could now go on to study increasingly complex examples of the kind as presented in the table above and they would clearly provide nice exercises for LEO-II+E. However, we do not expect that LEO-II+E can compete with specialist modal logic provers on challenge problems. In order to gain a highly efficient mechanization of challenge problems, we may therefore want to develop a cooperation with fast modal logic provers as well and pass the problems before expansion of the definitions from LEO-II directly to them. Verification of their results within LEO-II could then be tackled afterwards. Verification of the cooperative proofs is an open issue also in our current translational approach since the refutations of the prover E cannot yet be translated back into refutations in LEO-II. This is also on our list of future work.

In the next Section will turn our focus to a more interesting question: using LEO-II to reason *about* some properties of different modal logics, rather than to reason *within* such modal logics.

5 Exploring Properties of Λ_0^{mm} with LEO-II

We now study study questions such as, *'What is the concrete modal logic we have introduced?'* First, we turn our attention to the weakest modal logic, namely K. The essential properties of K are the necessitation rule N and the distribution axiom D (for all accessibility relations R and modal propositions A, B):

$$\begin{array}{ll} N & \textit{If } A \textit{ is a theorem of } K\textit{, then so is } \Box_R\, A \\ D & \Box_R\,(A \Rightarrow B) \Rightarrow (\Box_R\, A \Rightarrow \Box_R\, B) \end{array}$$

Our modal logic definitions in fact entail these principles and LEO-II can easily prove them: We formalize problem K_N, expressing that N is entailed by our definitions so far, as

$$\forall R_{\blacksquare} \forall A_{\blacksquare} \mathrm{valid}(A) \Rightarrow \mathrm{valid}(\Box_R\, A)$$

LEO-II+E can prove it easily in 0.027 seconds. Problem K_D, expressing that D is valid, is formalized as

$$\forall R_{\blacksquare}\forall A_{\blacksquare}\forall B_{\blacksquare}\mathrm{valid}(\Box_R\,(A \Rightarrow B) \Rightarrow (\Box_R\,A \Rightarrow \Box_R\,B))$$

LEO-II+E proves it in 0.029 seconds.

This provides some evidence that we have indeed correctly modeled the modal logic K in simple type theory. So let us call the system we developed so far $\boldsymbol{K}$. Modal logic $S4$ is obtained from modal logic K by adding the axioms T and 4. These axioms are well known to correspond to reflexivity and transitivity of the accessibility relation R.

$$\begin{array}{rll} T & \Box_R\,A \Rightarrow A & \textit{corresponds to}\text{: } \mathrm{refl}(R) \\ 4 & \Box_R\,A \Rightarrow \Box_R\,\Box_R\,A & \textit{corresponds to}\text{: } \mathrm{trans}(R) \end{array}$$

Reflexivity and transitivity are defined in the obvious manner:

$$\begin{aligned} \mathrm{refl} &:= \lambda R_{\blacksquare}\forall X_{\blacksquare}R\,X\,X \\ \mathrm{trans} &:= \lambda R_{\blacksquare}\forall X_{\blacksquare}\forall Y_{\blacksquare}\forall Z_{\blacksquare}R\,X\,Y \wedge R\,Y\,Z \Rightarrow R\,X\,Z. \end{aligned}$$

We will later also use irreflexivity and symmetry:

$$\begin{aligned} \mathrm{irrefl} &:= \lambda R_{\blacksquare}\forall X_{\blacksquare}\neg(R\,X\,X) \\ \mathrm{sym} &:= \lambda R_{\blacksquare}\forall X_{\blacksquare}\forall Y_{\blacksquare}R\,X\,Y \Rightarrow R\,Y\,X. \end{aligned}$$

We now study the following obvious questions *about* our system $\boldsymbol{K}$ within LEO-II:

K_T^a**:** *Is axiom T is valid in $\boldsymbol{K}$?* As expected, LEO-II cannot prove

$$\forall R_{\blacksquare}\forall A_{\blacksquare}\mathrm{valid}(\Box_R\,A \Rightarrow A).$$

K_T^b**:** *Is there a relation R such that for all modal propositions A, axiom T is valid in $\boldsymbol{K}$?* LEO-II+E can prove

$$\exists R_{\blacksquare}\forall A_{\blacksquare}\mathrm{valid}(\Box_R\,A \Rightarrow A)$$

in 0.456 seconds; the prover E is unsuccessfully called three times in LEO-II's reasoning loop before the refutation is finally detected within LEO-II itself. A clever instantiation for relation R is actually needed to solve this problem (as we know from our undergraduate modal logic course, R obviously needs to be reflexive) and this instantiation can neither be synthesized by E's first-order unification nor by

LEO-II's higher-order pre-unification. In fact, such an instantiation needs to be guessed by primitive substitution. LEO-II applies primitive substitution after the initial pre-processing phase and it proposes, amongst others, to consider the equality relation as a candidate instantiation for R. Then, LEO-II quickly finds the refutation. To be more precise, LEO-II does not instantiate R with $\lambda X_\iota . \lambda Y_\iota . X = Y$ in the crucial primitive substitution step but with the more general term $\lambda X_\iota . \lambda Y_\iota . (V\ X\ Y) = (W\ X\ Y)$ where V and W are new free variables. By instantiating V and W with projections on the first and second argument the former term could be introduced. However, as the proof protocol generated by LEO-II for this example problem in the Appendix B illustrates, the second, more general term already leads to a refutation (ending in pre-unification constraint consisting only in a flex-flex unification pair) without further instantiation of V or W.

For the exploration of the properties of modal logic ***K*** we have thus gained useful information: by considering the equality relation as an accessibility relation, which amongst other properties is reflexive, we are able to prove this statement. Thus, one way to proceed with the exploration is to investigate the connection between reflexivity and T in ***K***.

K_T^c**:** *Is axiom T indeed equivalent to reflexivity of R in* ***K****?* LEO-II+E takes 0.068 seconds to prove

$$\forall R . (\forall A . \mathrm{valid}(\Box_R\, A \Rightarrow A) \Leftrightarrow \mathrm{refl}(R)).$$

K_4^a**:** *Is axiom 4 valid in* ***K****?* As expected, LEO-II+E cannot prove

$$\forall R . \forall A . \mathrm{valid}(\Box_R\, A \Rightarrow \Box_R\, \Box_R\, A).$$

K_4^b**:** *Is there a relation R and a modal proposition A for which axiom 4 is valid in* ***K****?* LEO-II+E takes 0.055 seconds to prove

$$\exists R . \forall A . \mathrm{valid}(\Box_R\, A \Rightarrow \Box_R\, \Box_R\, A).$$

Interestingly, the relation generated for R is $\lambda X_\iota . \lambda Y_\iota . (V\ X\ Y) \neq (W\ X\ Y)$ where V and W are new free variables; see the proof protocol generated by LEO-II in the Appendix C. A simpler relation would be the empty relation $\lambda X_\iota . \lambda Y_\iota . X \neq X$ which is obviously an

instance of the above one.[3] We proceed with the exploration. Let us be naive this time and consider irreflexivity and symmetry as interesting properties first before investigating transitivity.

K_4^c: *Is axiom 4 equivalent to irreflexivity?* LEO-II+E cannot prove

$$\forall R_{\blacksquare}(\forall A_{\blacksquare}\,\mathrm{valid}(\Box_R\, A \Rightarrow \Box_R\, \Box_R\, A)) \Leftrightarrow \mathrm{irrefl}(R).$$

K_4^d: *Is axiom 4 equivalent to symmetry?* LEO-II+E cannot prove

$$\forall R_{\blacksquare}(\forall A_{\blacksquare}\,\mathrm{valid}(\Box_R\, A \Rightarrow \Box_R\, \Box_R\, A)) \Leftrightarrow \mathrm{sym}(R).$$

K_4^e: *Is axiom 4 equivalent to transitivity of R in $\boldsymbol{K}$?* LEO-II+E takes 0.193 seconds to prove

$$\forall R_{\blacksquare}(\forall A_{\blacksquare}\,\mathrm{valid}(\Box_R\, A \Rightarrow \Box_R\, \Box_R\, A)) \Leftrightarrow \mathrm{trans}(R),$$

K_{T4}^a: *Are axioms T and 4 equivalent to reflexivity and transitivity of R in $\boldsymbol{K}$?* LEO-II+E takes 2.329 seconds (of which LEO-II uses 0.164 seconds) to prove

$$\begin{aligned}\forall R_{\blacksquare}(\forall A_{\blacksquare}\,\mathrm{valid}(\Box_R\, A \Rightarrow A) \wedge \mathrm{valid}(\Box_R\, A \Rightarrow \Box_R\, \Box_R\, A))\\ \Leftrightarrow (\mathrm{refl}(R) \wedge \mathrm{trans}(R))\end{aligned}$$

E receives 70 clauses and generates 21769 before it finds the refutation. This example well illustrates the benefits of the cooperation between LEO-II and E, since the first-order refutation required in this example is already too challenging to be easily detected by LEO-II alone in its much more challenging search space.

It is much easier, however, to prove the two directions separately. LEO-II+E takes 0.040 seconds to prove

$$\begin{aligned}\forall R_{\blacksquare}(\forall A_{\blacksquare}\,\mathrm{valid}(\Box_R\, A \Rightarrow A) \wedge \mathrm{valid}(\Box_R\, A \Rightarrow \Box_R\, \Box_R\, A))\\ \Rightarrow (\mathrm{refl}(R) \wedge \mathrm{trans}(R))\end{aligned}$$

[3] The proof protocol in the Appendix C shows that from the initial problem statement LEO-II derives the clause 27 : $[V^1\ (sk^3 V^1)\ (sk^5 V^1)]^T$. It then instantiates V^1 with the term $t = \lambda V^9_{\blacksquare}\lambda V^{10}_{\blacksquare}((\lambda X_{\blacksquare}\lambda Y_{\blacksquare}\neg(X = Y))\ (V^{11}\ V^9\ V^{10})\ (V^{12}\ V^9\ V^{10}))$ by primitive substitution to obtain the clause 39 : $[\neg(V^{11}\ (sk^3\ t)\ (sk^5\ t)) = (V^{12}\ (sk^3\ t)\ (sk^5\ t))]^T$ and subsequently it generates the flex-flex unification constraint clause 59 : $[(V^{11}\ (sk^3\ t)\ (sk^5\ t)) = (V^{12}\ (sk^3\ t)\ (sk^5\ t))]^F$ by clause normalization. Flex-flex unification constraint clauses can always be refuted and thus proof search in LEO-II terminates here. For example, if we would instantiate V^{11} and V^{12} both with a term $\lambda X_{\blacksquare}\lambda Y_{\blacksquare}a$ for an arbitrary constant a then we would generate the the clause $[a = a]^F$ in which case the contradiction becomes obvious.

and 0.039 seconds for

$$\forall R_{\blacksquare}(\forall A_{\blacksquare}\,\mathrm{valid}(\Box_R\, A \Rightarrow A) \wedge \mathrm{valid}(\Box_R\, A \Rightarrow \Box_R\, \Box_R\, A)) \\ \Leftarrow (\mathrm{refl}(R) \wedge \mathrm{trans}(R)).$$

Thus, we have successfully explored the properties of modal logic $S4$ with LEO-II+E. And we could go on to explore properties of other more specialized modal logics and multi modal logics in exactly the same way.

6 Quantified Multimodal Logics Λ_1^{mm} and Λ_ω^{mm}

We adapt the definition of propositional multimodal logic to the first-order and higher-order case.

DEFINITION 3 (First-order Quantified Multimodal Logic Λ_1^{mm}). In addition to base type ι we introduce a second base type, μ. The idea is that ι is reserved to denote the set of all possible worlds while μ denotes the set of individuals. Let Σ be a signature.

1. Λ_1^{mm}-terms are defined as the smallest set of simply typed λ-terms for which the following hold:

   - Each constant $c_\mu \in \Sigma$ and variable $X_\mu \in \Sigma$ is a Λ_1^{mm}-term.
   - If $t_\mu^1, \ldots, t_\mu^n$ are Λ_1^{mm}-terms and $f_{\mu\to\ldots\to\mu\to\mu} \in \Sigma$ is an n-ary (curried) function symbol, then $(\ldots(f\, t^1)\ldots t^n)_\mu$ is a Λ_1^{mm}-term.

   Note that Λ_1^{mm}-terms must not depend on worlds, that is, subterms of a Λ_1^{mm}-term are never of type ι.

2. The modal operators $\neg$, $\vee$, $\Box_R$ are defined as before.

3. Modal universal quantification $\forall X_{\mu\,\blacksquare}\, \phi_{\iota\to o}$ is defined as the term

   $$\lambda w_{\iota\,\blacksquare} \forall X_{\mu\,\blacksquare} \phi\, w$$

   In fact we can employ the standard trick in simple type theory to avoid introducing a new binder for universal quantification. For this we introduce the logical constant $\mathbf{\Pi}_{\mu\to(\iota\to o)}$ and use this to encode modal universal quantification as follows: $\forall X_{\mu\,\blacksquare}\, \phi_{\iota\to o}$ stands for

   $$\mathbf{\Pi}_{\mu\to(\iota\to o)}(\lambda X_{\mu\,\blacksquare} \phi_{\iota\to o})$$

   and the modal operator $\mathbf{\Pi}$ is itself defined as

   $$\lambda \phi'_{\mu\to(\iota\to o)\,\blacksquare} \lambda W_{\iota\,\blacksquare} \forall X_{\mu\,\blacksquare} \phi'\, X\, W.$$

4. Λ_1^{mm}-propositions are defined as the smallest set of simply typed terms for which the following hold:
   - If $t_\mu^1, \ldots, t_\mu^n$ are Λ_1^{mm}-terms and $p_{\mu\to\ldots\to\mu\to(\iota\to o)} \in \Sigma$ is an n-ary (curried) predicate symbol (for $n \geq 0$), then $(\ldots(p\,t^1)\ldots t^n)_{\iota\to o}$ is an atomic Λ_1^{mm}-proposition.
   - If ϕ and ψ are Λ_1^{mm}-propositions, then so are $\neg\,\phi$, $\phi \vee \psi$ and $\Box_R\,\phi$, where R is a term of type $\iota \to \iota \to o$.
   - If $X_\mu \in \Sigma$ is a variable of type μ and $\phi_{\iota\to o}$ is a Λ_1^{mm}-proposition, then $\forall X_\cdot\,\phi$ is a Λ_1^{mm}-proposition.

First-order quantified multimodal logic has been studied in Nguyen [28], who defines a basic quantified normal multimodal logic called $K(m)$; like K in the propositional case, it serves as the starting point for the introduction of further quantified normal multimodal logics.

Here we briefly investigate whether Λ_1^{mm} fulfill the characteristics of the normal multimodal logic $K(m)$.

CLAIM 4 (Λ_1^{mm} is a normal multimodal logic). Λ_1^{mm} fulfill the properties of the normal multimodal logic $K(m)$ of [28]. That is:

1. Λ_1^{mm} validates the axioms for classical predicate logic.
2. Λ_1^{mm} validates the K-axioms: $\Box_R\,(\phi \Rightarrow \psi) \Rightarrow (\Box_R\,\phi \Rightarrow \Box_R\,\psi)$.
3. Λ_1^{mm} validates the Barcan formula axioms: $(\forall X_\cdot\,\Box_R\,\phi) \Rightarrow \Box_R\,\forall X_\cdot\,\phi$.
4. Λ_1^{mm} validates the axioms defining $\Diamond_R$: $\Diamond_R\,\phi \Leftrightarrow \neg\Box_R\,\neg\phi$.
5. Λ_1^{mm} validates the modus ponens rule: if ϕ and $\phi \Rightarrow \psi$ are valid, then ψ is valid.
6. Λ_1^{mm} validates the generalization rule: if ϕ is valid, then $\forall X_\cdot\,\phi$ is valid.
7. Λ_1^{mm} validates the modal generalization rule: if ϕ is valid, then $\Box_R\,\phi$ is valid.
8. $K(M)$ validates the converse Barcan formula $(\forall X_\cdot\,\Box_R\,\phi) \Leftarrow \Box_R\,\forall X_\cdot\,\phi$.

We do not give a formal proof of this claim here and instead argue as follows:

1. The axioms for classical predicate logic are obviously valid in simple type theory (and Leo-II can handle them).

2. LEO-II+E can prove $\forall R_{\blacksquare}\forall \phi_{\blacksquare}\forall \psi_{\blacksquare}\mathrm{valid}(\Box_R\,(\phi \Rightarrow \psi) \Rightarrow (\Box_R\,\phi \Rightarrow \Box_R\,\psi)$ in 0.059 seconds.

3. LEO-II+E can prove $\forall R_{\blacksquare}\forall \phi_{\blacksquare}\mathrm{valid}(\forall X_{\blacksquare}\,\Box_R\,\phi \Rightarrow \Box_R\,\forall X_{\blacksquare}\,\phi)$ in 0.066 seconds.

4. LEO-II+E can prove $\forall R_{\blacksquare}\forall \phi_{\blacksquare}\mathrm{valid}(\Diamond_R\,\phi \Leftrightarrow \neg\,\Box_R\,\neg\phi)$ in 0.078 seconds.

5. LEO-II+E can prove $\forall \phi_{\blacksquare}\forall \psi_{\blacksquare}\mathrm{valid}(\phi) \wedge \mathrm{valid}(\phi \Rightarrow \psi) \Rightarrow \mathrm{valid}(\psi)$ in 0.062 seconds.

6. LEO-II+E can prove

$$\forall P_{\mu\to(\iota\to o)\,\blacksquare}(\forall X_{\mu\,\blacksquare}\mathrm{valid}(P\,X)) \Rightarrow \mathrm{valid}(\forall X_{\mu\,\blacksquare}\,(P\,X)$$

   in 0.061 seconds.

7. LEO-II+E can prove $\forall R_{\blacksquare}\forall \phi_{\blacksquare}\mathrm{valid}(\phi) \Rightarrow \mathrm{valid}(\Box_R\,\phi)$ in 0.056 seconds.

8. LEO-II+E can prove $\Box_R\,\forall X_{\blacksquare}\,\phi \Rightarrow \forall X_{\blacksquare}\,\Box_R\,\phi$, the converse Barcan formula, in 0.049 seconds.

Assuming that LEO-II+E is correct, this provides evidence that Λ_1^{mm} indeed adequately models the basic normal multimodal logic $K(m)$ of Nguyen [28]. Validity of the Barcan formula and its converse illustrate that quantification in Λ_1^{mm} is a fixed-domain quantification.

The first-order quantified multimodal logic Λ_1^{mm} can be further generalized. Doing so we obtain the following proposal for a higher order quantified multimodal logic Λ_ω^{mm}, again with fixed-domain quantification.

DEFINITION 5 (Higher-order Quantified Multimodal Logic Λ_ω^{mm}). The key idea of this definition is to exclude statements about possible worlds from the language Λ_ω^{mm}. The rest is as before.

1. Given a simply typed λ-term t_γ we define the function *obt* (occurring base types) as follows:

   - $obt(t = c_\delta \in \Sigma) = obt'(\delta)$
   - $obt(t = (t^1\,t^2)) = obt(t^1) \cup obt(t^2)$
   - $obt(t = (\lambda X_{\blacksquare} t^1)) = obt(X) \cup obt(t^1)$
   - $obt'(\delta \in \mathcal{BT}) = \{\delta\}$
   - $obt'(\beta \to \delta) = obt'(\beta) \cup obt'(\delta)$

Let $t_{\iota\to o}$ be a simply typed λ-term. If $t_{\iota\to o}$ is a constant or variable, then $t_{\iota\to o}$ is a Λ^{mm}_{ω}-term. If $t_{\iota\to o}$ is an abstraction $\lambda X_{\iota}\,.\,p_o$, then $t_{\iota\to o}$ is a Λ^{mm}_{ω}-term if $\iota \notin \mathit{obt}(p_o)$. If t is an application $(q_{\alpha\to(\iota\to o))}\ s_{\alpha})$, then $t_{\iota\to o}$ is a Λ^{mm}_{ω}-term if $\iota \notin \mathit{obt}(s_{\alpha})$.

2. The modal operators $\neg$, $\vee$, $\Box_R$ are defined as before.

3. Modal universal quantification $\forall X_{\gamma}\,.\,\phi_{\iota\to o}$ is now defined follows: Let $\phi_{\iota\to o}$ be an Λ^{mm}_{ω}-term and let $X_{\gamma} \in \Sigma$ be a variable such that $\iota \notin \mathit{obt}(X_{\gamma})$. Then $\forall X_{\gamma}\,.\,\phi$ stands for $\lambda w_{\iota}\,.\,\forall X_{\gamma}\,.\,\phi\, w$.

   Again we can employ the standard trick in simple type theory to avoid introducing a new binder for universal quantification. This time we introduce the logical constants $\mathbf{\Pi}_{\gamma\to(\iota\to o)}$ for all types γ such that $\iota \notin \mathit{obt}'(\gamma)$. We use them to encode modal universal quantification as follows: $\forall X_{\gamma}\,.\,\phi_{\iota\to o}$ stands for $\mathbf{\Pi}_{\gamma\to(\iota\to o)}(\lambda X_{\gamma}\,.\,\phi_{\iota\to o})$ and the modal operator $\mathbf{\Pi}$ is itself defined as

$$\lambda\phi'_{\gamma\to(\iota\to o)}\,.\,\lambda W_{\iota}\,.\,\forall X_{\gamma}\,.\,\phi'\, X\, W.$$

4. Λ^{mm}_{ω}-propositions are defined as the smallest set of simply typed λ-terms for which the following hold:

   - Each Λ^{mm}_{ω}-term $t_{\iota\to o}$ is an atomic Λ^{mm}_{ω}-proposition.
   - If ϕ and ψ are Λ^{mm}_{ω}-propositions, then so are $\neg\,\phi$, $\phi \vee \psi$ and $\Box_R\,\phi$.
   - If $X_{\gamma} \in \Sigma$ is a variable of type μ and $\phi_{\iota\to o}$ is a Λ^{mm}_{ω}-proposition, then $\forall X\,.\,\phi$ is a Λ^{mm}_{ω}-proposition.

We could now proceed with a systematic exploration of the properties of Λ^{mm}_{1} and Λ^{mm}_{ω}. This, however, will be future work.

7 Discussion

In this paper we explore an interesting and promising research direction: the embedding of multimodal logic in simple type theory, the development of reasoning tools such as Leo-II or TPS [5] to support reasoning in and about multimodal logics in simple type theory, and the systematic computer supported exploration of properties of multimodal logic in simple type theory.

Interesting future work will be to employ the presented encoding of normal multimodal logics in order to attack the \$100 modal logic challenge[4]

[4] `http://www.cs.miami.edu/~tptp/HHDC/`

with LEO-II. This challenge, originally proposed by John Halleck, calls for a computer program, which given the formalizations of any two modal logics determines their relationship. The approach presented in this paper should generally be applicable to this challenge when restricting it to *normal* multimodal logics.

We illustrate the idea by an example. Consider the statement

$$\exists R_{\blacksquare} \exists A_{\blacksquare} \exists B_{\blacksquare} (\neg \text{valid}(\Box_R A \Rightarrow A)) \vee (\neg \text{valid}(\Box_R B \Rightarrow \Box_R \Box_R B))$$

LEO-II alone (without E) can prove this statement in 17.305 seconds.[5] Using primitive substitution, LEO-II instantiates R in this proof with the "not equals" relation, which is non-reflexive. We can conclude from this proof that modal logic $S4$ is not entailed by modal logic K ($S4 \not\subseteq K$).

The time of 17.305 seconds required by LEO-II (without any tuning) is clearly a massive improvement over the 16 minutes pre-processing time plus 2710 seconds solving time reported by Rabe et al. [31] for the same problem. They employ a specifically tuned system based on standard first-order theorem provers and standard translations of modal logics into first-order logic.

However, in order to solve the challenge for all normal multimodal logics, a specific tuning of LEO-II seems also unavoidable. The problem is related to primitive substitution. It is well illustrated by the following statement, which expresses that modal logic T (which adds only axiom T to K) is not entailed in K:

$$\exists R_{\blacksquare} \exists B_{\blacksquare} (\neg \text{valid}(\Box_R B \Rightarrow \Box_R \Box_R B))$$

We would expect that this statement can be quickly proved when instantiating R with a non-transitive relation similar to primitive substitution steps required in examples K^b_T and K^b_4. For good reasons LEO-II fails to do so. In fact, LEO-II also fails to prove the related statement

$$\exists R_{\blacksquare} \neg \text{trans}(R)$$

The reason is that without further assumptions this statement is not a theorem. We have neither assumed the axiom of infinity nor that there exist at least two different possible worlds. Hence, our domain of possible worlds may well just consist of a single world w in which case a non-transitive accessibility relation cannot be provided.

[5] The proof in LEO-II+E takes 198.818 seconds because of several unsuccessful but time consuming calls to E. We have already mentioned that the current sequential architecture of LEO-II+E needs to be replaced by a distributed in which LEO-II cooperates with the first-order prover in a distributed and incremental manner.

LEO-II can in fact prove the existence of a non-transitive accessibility relation under the additional assumption

$$\exists X . \exists Y . X \neq Y$$

Like the special purpose approach used by Rabe et al. [31], we could probably successfully tune LEO-II to provide and employ a domain specific "menu" of the interesting accessibility relations in connection with interesting constraints on the domain of possible worlds in order to attack the $100 modal logic challenge.

Acknowledgment:

We thank Chad Brown, Franz Baader, Gert Smolka, and Mark Kaminski for their valuable comments to earlier versions of this paper and John Harrison for reminding us of this interesting application area. We also thank Catalin Hritcu for proofreading.

BIBLIOGRAPHY

[1] Peter B. Andrews. General Models and Extensionality. *Journal of Symbolic Logic*, 37:395–397, 1972.

[2] Peter B. Andrews. General Models, Descriptions, and Choice in Type Theory. *Journal of Symbolic Logic*, 37:385–394, 1972.

[3] Peter B. Andrews. On Connections and Higher-Order Logic. *Journal of Automated Reasoning*, 5:257–291, 1989.

[4] Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. Kluwer Academic Publishers, second edition, 2002.

[5] Peter B. Andrews and Chad E. Brown. TPS: A hybrid automatic-interactive system for developing proofs. *J. Applied Logic*, 4(4):367–395, 2006.

[6] Philippe Balbiani, Luis Fariñas del Cerro, and Andreas Herzig. Declarative Semantics for Modal Logic Programs. In *FGCS*, pages 507–514, 1988.

[7] Matteo Baldoni, Laura Giordano, and Alberto Martelli. A Framework for a Modal Logic Programming. In *Joint International Conference and Symposium on Logic Programming*, pages 52–66, 1996.

[8] Christoph Benzmüller, Chad E. Brown, and Michael Kohlhase. Higher Order Semantics and Extensionality. *Journal of Symbolic Logic*, 69:1027–1088, 2004.

[9] Christoph Benzmüller, Larry Paulson, Frank Theiss, and Arnaud Fietzke. Progress Report on LEO-II – An Automatic Theorem Prover for Higher-Order Logic. In *Emerging Trends at the 20th International Conference on Theorem Proving in Higher Order Logics*. University Kaiserslautern, Germany, 2007.

[10] Christoph Benzmüller, Volker Sorge, Mateja Jamnik, and Manfred Kerber. Combined Reasoning by Automated Cooperation. *Journal of Applied Logic*, 2007. To appear.

[11] Chad E. Brown. Encoding Hybrid Logic in Higher-Order Logic. Unpublished slides from an invited talk presented at Loria Nancy, France, April 2005. `http://mathgate.info/cebrown/papers/hybrid-hol.pdf`.

[12] Bob Carpenter. *Type-logical semantics*. MIT Press, Cambridge, MA, USA, 1998.

[13] Alonzo Church. A Formulation of the Simple Theory of Types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[14] Luis Fariñas del Cerro. MOLOG: A System That Extends PROLOG with Modal Logic. *New Generation Comput.*, 4(1):35–50, 1986.
[15] Daniel Gallin. *Intensional and Higher-Order Modal Logic*, volume 19 of *North-Holland Mathematics Studies*. North-Holland, Amsterdam, 1975.
[16] L. T. F. Gamut. *Logic, Language, and Meaning. Volume II. Intensional Logic and Logical Grammar*, volume 2. The University of Chicago Press, 1991.
[17] Moritz Hardt and Gert Smolka. Higher-Order Syntax and Saturation Algorithms for Hybrid Logic. *Electr. Notes Theor. Comput. Sci.*, 174(6):15–27, 2007.
[18] John Harrison. *HOL Light Tutorial (for version 2.20)*. Intel JF1-13, September 2006. `http://www.cl.cam.ac.uk/~jrh13/hol-light/tutorial_220.pdf`.
[19] Leon Henkin. Completeness in the Theory of Types. *Journal of Symbolic Logic*, 15:81–91, 1950.
[20] Gérard P. Huet. A Mechanization of Type Theory. In *Proceedings of the Third International Joint Conference on Artificial Intelligence*, pages 139–146, Stanford University, California, USA, 1973. IJCAI.
[21] J. Hurd. An LCF-Style Interface between HOL and First-Order Logic. In *Automated Deduction — CADE-18*, volume 2392 of *LNAI*, pages 134–138. Springer, 2002.
[22] Mark Kaminski and Gert Smolka. Hybrid Tableax for the Difference Modality, 2007. Accepted to Methods for Modalities 5 (M4M-5), Cachan, France.
[23] Manfred Kerber. *On the Representation of Mathematical Concepts and their Translation into First-Order Logic*. PhD thesis, Fachbereich Informatik, Universität Kaiserslautern, Kaiserslautern, Germany, 1992.
[24] Jia Meng and Lawrence C. Paulson. Experiments on Supporting Interactive Proof Using Resolution. In *In Proc. of IJCAR 2004*, volume 3097 of *LNCS*, pages 372–384. Springer, 2004.
[25] Jia Meng, Claire Quigley, and Lawrence C. Paulson. Automation for interactive proof: first prototype. *Inf. Comput.*, 204(10):1575–1596, 2006.
[26] Stephan Merz. Yet another encoding of TLA in Isabelle. Available on the Internet: `http://www.loria.fr/~merz/projects/isabelle-tla/doc/design.ps.gz`, 1999.
[27] Linh Anh Nguyen. A Fixpoint Semantics and an SLD-Resolution Calculus for Modal Logic Programs. *Fundam. Inform.*, 55(1):63–100, 2003.
[28] Linh Anh Nguyen. Multimodal logic programming. *Theor. Comput. Sci.*, 360(1-3):247–288, 2006.
[29] Andreas Nonnengart. How to Use Modalities and Sorts in Prolog. In Craig MacNish, David Pearce, and Luís Moniz Pereira, editors, *JELIA*, volume 838 of *Lecture Notes in Computer Science*, pages 365–378. Springer, 1994.
[30] Hans Jürgen Ohlbach. A Resolution Calculus for Modal Logics. In Ewing L. Lusk and Ross A. Overbeek, editors, *CADE*, volume 310 of *Lecture Notes in Computer Science*, pages 500–516. Springer, 1988.
[31] Florian Rabe, Petr Pudlak, Geoff Sutcliffe, and Weina Shen. Solving the $100 modal logic challenge. *Journal of Applied Logic*, 2007.
[32] Stephan Schulz. E – A Brainiac Theorem Prover. *Journal of AI Communications*, 15(2/3):111–126, 2002.

A LEO-II Proof Protocol for Example K^d

```
**** Protocol for Problem: K^d.thf
**** Beginning of derivation protocol ****
1: (mvalid @ ((mbox @ s) @ ((mimpl @ ((mbox @ r) @ a)) @
 ((mbox @ r) @ a))))=$true
 ---  theorem(file(K^d.thf,[thm]))
2: (mvalid @ ((mbox @ s) @ ((mimpl @ ((mbox @ r) @ a)) @
 ((mbox @ r) @ a))))=$false
 ---  neg_input 1
```

```
3: (! [x0:$i,x1:$i] : ((~ ((s @ x0) @ x1)) | ((~ (! [x2:$i] :
 ((~ ((r @ x1) @ x2)) | (a @ x2)))) | (! [x2:$i] :
 ((~ ((r @ x1) @ x2)) | (a @ x2))))))=$false
 ---  unfold_def 2
4: (! [x1:$i] : ((~ ((s @ sk1) @ x1)) | ((~ (! [x2:$i] :
 ((~ ((r @ x1) @ x2)) | (a @ x2)))) | (! [x2:$i] :
 ((~ ((r @ x1) @ x2)) | (a @ x2))))))=$false
 ---  cnf 3
5: ((~ ((s @ sk1) @ sk2)) | ((~ (! [x2:$i] :
 ((~ ((r @ sk2) @ x2)) | (a @ x2)))) | (! [x2:$i] :
 ((~ ((r @ sk2) @ x2)) | (a @ x2)))))=$false
 ---  cnf 4
6: ((~ (! [x2:$i] : ((~ ((r @ sk2) @ x2)) | (a @ x2)))) |
 (! [x2:$i] : ((~ ((r @ sk2) @ x2)) | (a @ x2))))=$false
 ---  cnf 5
7: (~ ((s @ sk1) @ sk2))=$false
 ---  cnf 5
8: ((s @ sk1) @ sk2)=$true
 ---  cnf 7
9: (! [x2:$i] : ((~ ((r @ sk2) @ x2)) | (a @ x2)))=$false
 ---  cnf 6
10: (~ (! [x2:$i] : ((~ ((r @ sk2) @ x2)) | (a @ x2))))=$false
 ---  cnf 6
11: (! [x2:$i] : ((~ ((r @ sk2) @ x2)) | (a @ x2)))=$true
 ---  cnf 10
12: ((~ ((r @ sk2) @ sk3)) | (a @ sk3))=$false
 ---  cnf 9
13: ((~ ((r @ sk2) @ V1)) | (a @ V1))=$true
 ---  cnf 11
14: (a @ sk3)=$false
 ---  cnf 12
15: (~ ((r @ sk2) @ sk3))=$false
 ---  cnf 12
16: (~ ((r @ sk2) @ V1))=$true | (a @ V1)=$true
 ---  cnf 13
17: ((r @ sk2) @ sk3)=$true
 ---  cnf 15
18: ((r @ sk2) @ V1)=$false | (a @ V1)=$true
 ---  cnf 16
19: ($false)=$true
 ---  fo-atp 8 14 17 18
**** End of derivation protocol ****
**** no. of clauses: 19 ****
```

B LEO-II Proof Protocol for Example K_T^b

```
**** Protocol for Problem: K_T^b.thf
**** Beginning of derivation protocol ****
1: (? [R:$i>($i>$o)] : (! [A:$i>$o] :
 (mvalid @ ((mimpl @ ((mbox @ R) @ A)) @ A))))=$true
 ---  theorem(file(K_T^b.thf,[thm]))
2: (? [R:$i>($i>$o)] : (! [A:$i>$o] :
 (mvalid @ ((mimpl @ ((mbox @ R) @ A)) @ A))))=$false
```

```
 ---  neg_input 1
3: (~ (! [x0:$i>($i>$o)] : (~ (! [x1:$i>$o,x2:$i] :
 ((~ (! [x3:$i] : ((~ ((x0 @ x2) @ x3)) | (x1 @ x3)))) |
 (x1 @ x2))))))=$false
 ---  unfold_def 2
4: (! [x0:$i>($i>$o)] : (~ (! [x1:$i>$o,x2:$i] :
 ((~ (! [x3:$i] : ((~ ((x0 @ x2) @ x3)) | (x1 @ x3)))) |
 (x1 @ x2)))))=$true
 ---  cnf 3
5: (~ (! [x1:$i>$o,x2:$i] : ((~ (! [x3:$i] : ((~ ((V1 @ x2) @ x3)) |
 (x1 @ x3)))) | (x1 @ x2))))=$true
 ---  cnf 4
6: (! [x1:$i>$o,x2:$i] : ((~ (! [x3:$i] :
 ((~ ((V1 @ x2) @ x3)) | (x1 @ x3)))) | (x1 @ x2)))=$false
 ---  cnf 5
7: (! [x2:$i] : ((~ (! [x3:$i] : ((~ ((V1 @ x2) @ x3)) |
 ((sk1 @ V1) @ x3)))) | ((sk1 @ V1) @ x2)))=$false
 ---  cnf 6
8: ((~ (! [x3:$i] : ((~ ((V1 @ (sk2 @ V1)) @ x3)) |
 ((sk1 @ V1) @ x3)))) | ((sk1 @ V1) @ (sk2 @ V1)))=$false
 ---  cnf 7
9: ((sk1 @ V1) @ (sk2 @ V1))=$false
 ---  cnf 8
10: (~ (! [x3:$i] : ((~ ((V1 @ (sk2 @ V1)) @ x3)) |
 ((sk1 @ V1) @ x3))))=$false
 ---  cnf 8
11: (! [x3:$i] : ((~ ((V1 @ (sk2 @ V1)) @ x3)) |
 ((sk1 @ V1) @ x3)))=$true
 ---  cnf 10
12: ((~ ((V1 @ (sk2 @ V1)) @ V2)) | ((sk1 @ V1) @ V2))=$true
 ---  cnf 11
13: (~ ((V1 @ (sk2 @ V1)) @ V2))=$true | ((sk1 @ V1) @ V2)=$true
 ---  cnf 12
14: ((V1 @ (sk2 @ V1)) @ V2)=$false | ((sk1 @ V1) @ V2)=$true
 ---  cnf 13
19: (((V19 @ (sk2 @ (^ [x0:$i,x1:$i] :
 (((V19 @ x0) @ x1) = ((V20 @ x0) @ x1))))) @ V2) =
 ((V20 @ (sk2 @ (^ [x0:$i,x1:$i] : (((V19 @ x0) @ x1) =
 ((V20 @ x0) @ x1))))) @ V2))=$false |
 ((sk1 @ (^ [x0:$i,x1:$i] : (((V19 @ x0) @ x1) =
 ((V20 @ x0) @ x1)))) @ V2)=$true
 ---  prim-subst (V1-->lambda [V17]: lambda [V18]:
 (eq ((V19 V17) V18)) ((V20 V17) V18)) 14
31: ((sk1 @ (^ [x0:$i,x1:$i] : (((V19 @ x0) @ x1) =
 ((V20 @ x0) @ x1)))) @ V2)=$true |
 (((V19 @ (sk2 @ (^ [x0:$i,x1:$i] : (((V19 @ x0) @ x1) =
 ((V20 @ x0) @ x1))))) @ V2) = ((V20 @ (sk2 @ (^ [x0:$i,x1:$i] :
 (((V19 @ x0) @ x1) = ((V20 @ x0) @ x1))))) @ V2))=$false
 ---  uni () 19
32: ((sk1 @ V32) @ (sk2 @ V32))=$false
 ---  rename 9
94: (((sk1 @ (^ [x0:$i,x1:$i] : (((V19 @ x0) @ x1) =
```

```
 ((V20 @ x0) @ x1)))) @ V2) = ((sk1 @ V32) @ (sk2 @ V32)))=$false |
 (((V19 @ (sk2 @ (^ [x0:$i,x1:$i] : (((V19 @ x0) @ x1) =
 ((V20 @ x0) @ x1))))) @ V2) = ((V20 @ (sk2 @ (^ [x0:$i,x1:$i] :
 (((V19 @ x0) @ x1) = ((V20 @ x0) @ x1))))) @ V2))=$false
 ---  res 31 32
97: (((V64 @ ((((sk9 @ (sk2 @ (^ [x0:$i,x1:$i] : (((V64 @ x0) @ x1) =
 ((V65 @ x0) @ x1))))) @ (^ [x0:$i,x1:$i] : (((V64 @ x0) @ x1) =
 ((V65 @ x0) @ x1)))) @ V20) @ V19)) @ ((((sk10 @ V19) @ V20) @
 (sk2 @ (^ [x0:$i,x1:$i] : (((V64 @ x0) @ x1) = ((V65 @ x0) @ x1))))) @
 (^ [x0:$i,x1:$i] : (((V64 @ x0) @ x1) = ((V65 @ x0) @ x1))))) =
 ((V65 @ ((((sk9 @ (sk2 @ (^ [x0:$i,x1:$i] : (((V64 @ x0) @ x1) =
 ((V65 @ x0) @ x1))))) @ (^ [x0:$i,x1:$i] : (((V64 @ x0) @ x1) =
 ((V65 @ x0) @ x1)))) @ V20) @ V19)) @ ((((sk10 @ V19) @ V20) @
 (sk2 @ (^ [x0:$i,x1:$i] : (((V64 @ x0) @ x1) = ((V65 @ x0) @ x1))))) @
 (^ [x0:$i,x1:$i] : (((V64 @ x0) @ x1) = ((V65 @ x0) @ x1)))))=$false |
 (((V19 @ ((((sk9 @ (sk2 @ (^ [x0:$i,x1:$i] : (((V64 @ x0) @ x1) =
 ((V65 @ x0) @ x1))))) @ (^ [x0:$i,x1:$i] : (((V64 @ x0) @ x1) =
 ((V65 @ x0) @ x1)))) @ V20) @ V19)) @ ((((sk10 @ V19) @ V20) @
 (sk2 @ (^ [x0:$i,x1:$i] : (((V64 @ x0) @ x1) = ((V65 @ x0) @ x1))))) @
 (^ [x0:$i,x1:$i] : (((V64 @ x0) @ x1) = ((V65 @ x0) @ x1))))) =
 ((V20 @ ((((sk9 @ (sk2 @ (^ [x0:$i,x1:$i] : (((V64 @ x0) @ x1) =
 ((V65 @ x0) @ x1))))) @ (^ [x0:$i,x1:$i] : (((V64 @ x0) @ x1) =
 ((V65 @ x0) @ x1)))) @ V20) @ V19)) @ ((((sk10 @ V19) @ V20) @ (sk2 @
 (^ [x0:$i,x1:$i] : (((V64 @ x0) @ x1) = ((V65 @ x0) @ x1))))) @
 (^ [x0:$i,x1:$i] : (((V64 @ x0) @ x1) = ((V65 @ x0) @ x1)))))=$false |
 (((V19 @ (sk2 @ (^ [x0:$i,x1:$i] : (((V19 @ x0) @ x1) =
 ((V20 @ x0) @ x1))))) @ (sk2 @ (^ [x0:$i,x1:$i] : (((V64 @ x0) @ x1) =
 ((V65 @ x0) @ x1))))) = ((V20 @ (sk2 @ (^ [x0:$i,x1:$i] :
 (((V19 @ x0) @ x1) = ((V20 @ x0) @ x1))))) @ (sk2 @ (^ [x0:$i,x1:$i] :
 (((V64 @ x0) @ x1) = ((V65 @ x0) @ x1)))))=$false
 ---  uni ( V32/lambda [V62]: lambda [V63]: (eq ((V64 V62) V63))
 ((V65 V62) V63)  V2/sk2 (lambda [x0]: lambda [x1]: (eq ((V64 x0) x1))
 ((V65 x0) x1)) ) 94
98: ($false)=$true  ---  flexflex 97
**** End of derivation protocol ****
**** no. of clauses: 20 ****
```

C LEO-II Proof Protocol for Example K_4^b

```
**** Protocol for Problem: K_4^b.thf
**** Beginning of derivation protocol ****
2: (? [R:$i>($i>$o)] : (! [A:$i>$o] : (mvalid @ ((mimpl @ ((mbox @ R) @ A))
 @ ((mbox @ R) @ ((mbox @ R) @ A))))))=$true
 ---  theorem(file(K_4^b.thf,[thm]))
4: (? [R:$i>($i>$o)] : (! [A:$i>$o] : (mvalid @ ((mimpl @ ((mbox @ R) @ A))
 @ ((mbox @ R) @ ((mbox @ R) @ A))))))=$false
 ---  neg_input 2
5: (~ (! [x0:$i>($i>$o)] : (~ (! [x1:$i>$o,x2:$i] : ((~ (! [x3:$i] :
 ((~ ((x0 @ x2) @ x3)) | (x1 @ x3)))) | (! [x3:$i] : ((~ ((x0 @ x2) @ x3))
 | (! [x4:$i] : ((~ ((x0 @ x3) @ x4)) | (x1 @ x4)))))))))=$false
 ---  unfold_def 4
7: (! [x0:$i>($i>$o)] : (~ (! [x1:$i>$o,x2:$i] : ((~ (! [x3:$i] :
 ((~ ((x0 @ x2) @ x3)) | (x1 @ x3)))) | (! [x3:$i] : ((~ ((x0 @ x2) @ x3))
```

```
 | (! [x4:$i] : ((~ ((x0 @ x3) @ x4)) | (x1 @ x4)))))))))=$true
 ---  cnf 5
9: (~ (! [x1:$i>$o,x2:$i] : ((~ (! [x3:$i] : ((~ ((V1 @ x2) @ x3)) |
 (x1 @ x3)))) | (! [x3:$i] : ((~ ((V1 @ x2) @ x3)) | (! [x4:$i] :
 ((~ ((V1 @ x3) @ x4)) | (x1 @ x4)))))))=$true
 ---  cnf 7
11: (! [x1:$i>$o,x2:$i] : ((~ (! [x3:$i] : ((~ ((V1 @ x2) @ x3)) |
 (x1 @ x3)))) | (! [x3:$i] : ((~ ((V1 @ x2) @ x3)) | (! [x4:$i] :
 ((~ ((V1 @ x3) @ x4)) | (x1 @ x4)))))))=$false
 ---  cnf 9
13: (! [x2:$i] : ((~ (! [x3:$i] : ((~ ((V1 @ x2) @ x3)) |
 ((sk2 @ V1) @ x3)))) | (! [x3:$i] : ((~ ((V1 @ x2) @ x3)) | (! [x4:$i] :
 ((~ ((V1 @ x3) @ x4)) | ((sk2 @ V1) @ x4)))))))=$false
 ---  cnf 11
15: ((~ (! [x3:$i] : ((~ ((V1 @ (sk3 @ V1)) @ x3)) | ((sk2 @ V1) @ x3)))) |
 (! [x3:$i] : ((~ ((V1 @ (sk3 @ V1)) @ x3)) | (! [x4:$i] :
 ((~ ((V1 @ x3) @ x4)) | ((sk2 @ V1) @ x4))))))=$false
 ---  cnf 13
17: (! [x3:$i] : ((~ ((V1 @ (sk3 @ V1)) @ x3)) | (! [x4:$i] :
 ((~ ((V1 @ x3) @ x4)) | ((sk2 @ V1) @ x4)))))=$false
 ---  cnf 15
21: ((~ ((V1 @ (sk3 @ V1)) @ (sk5 @ V1))) | (! [x4:$i] :
 ((~ ((V1 @ (sk5 @ V1)) @ x4)) | ((sk2 @ V1) @ x4))))=$false
 ---  cnf 17
25: (~ ((V1 @ (sk3 @ V1)) @ (sk5 @ V1)))=$false
 ---  cnf 21
27: ((V1 @ (sk3 @ V1)) @ (sk5 @ V1))=$true
 ---  cnf 25
39: (~ (((V11 @ (sk3 @ (^ [x0:$i,x1:$i] : (~ (((V11 @ x0) @ x1) =
 ((V12 @ x0) @ x1)))))) @ (sk5 @ (^ [x0:$i,x1:$i] : (~ (((V11 @ x0) @ x1) =
 ((V12 @ x0) @ x1)))))) = ((V12 @ (sk3 @ (^ [x0:$i,x1:$i] :
 (~ (((V11 @ x0) @ x1) = ((V12 @ x0) @ x1)))))) @ (sk5 @ (^ [x0:$i,x1:$i] :
 (~ (((V11 @ x0) @ x1) = ((V12 @ x0) @ x1)))))))=$true
 ---  prim-subst (V1-->lambda [V9]: lambda [V10]: ((lambda [X]: lambda [Y]:
 neg ((eq X) Y)) ((V11 V9) V10)) ((V12 V9) V10)) 27
59: (((V11 @ (sk3 @ (^ [x0:$i,x1:$i] : (~ (((V11 @ x0) @ x1) =
 ((V12 @ x0) @ x1)))))) @ (sk5 @ (^ [x0:$i,x1:$i] : (~ (((V11 @ x0) @ x1) =
 ((V12 @ x0) @ x1)))))) = ((V12 @ (sk3 @ (^ [x0:$i,x1:$i] :
 (~ (((V11 @ x0) @ x1) = ((V12 @ x0) @ x1)))))) @ (sk5 @ (^ [x0:$i,x1:$i] :
 (~ (((V11 @ x0) @ x1) = ((V12 @ x0) @ x1)))))))=$false
 ---  cnf 39
60: ($false)=$true  ---  flexflex 59
**** End of derivation protocol ****
**** no. of clauses: 15 ****
```