Verified Decision Procedures for Modal Logics

Minchao Wu
Research School of Computer Science, Australian National University, Australia

Rajeev Goré
Research School of Computer Science, Australian National University, Australia

Abstract

We describe a formalization of modal tableaux with histories for the modal logics K, KT and S4 in Lean. We describe how we formalized the static and transitional rules, the non-trivial termination and the correctness of loop-checks. The formalized tableaux are essentially executable decision procedures with soundness and completeness proved. Termination is also proved in order to define them as functions in Lean. All of these decision procedures return a concrete Kripke model in cases where the input set of formulas is satisfiable, and a proof constructed via the tableau rules witnessing unsatisfiability otherwise. We also describe an extensible formalization of backjumping and its verified implementation for the modal logic K. As far as we know, these are the first verified decision procedures for these modal logics.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Type theory; Software and its engineering → Formal methods

Keywords and phrases Formal Methods, Interactive Theorem Proving, Modal Logic, Lean

Digital Object Identifier 10.4230/LIPIcs.ITP.2019.31

Supplement Material The formalization is available at `https://github.com/minchaowu/ModalTab`

1 Introduction

Propositional modal logics have proved useful for reasoning about knowledge and belief [24], verifying digital circuits [8], and knowledge representation and reasoning [1].

The main reason for their success is that they provide just the right amount of extra expressive power, somewhere between propositional and first order logic, while retaining almost universal decidability. Modal description logics [3] in particular are extremely expressive, many with decision procedures that are EXPTIME-complete and beyond.

There are many efficient implementations of various modal and description logics, but the desire for efficiency leads to numerous non-trivial optimizations which make the theoretical soundness and completeness harder to prove. Consequently, most of these implementations are buggy and require constant maintenance to iron out these bugs. For efficiency, these provers also do not provide concrete evidence, such as proofs or countermodels, for their answers. As such, these implementations cannot be used in safety-critical applications.

Naive decision procedures for modal logics sometimes proceed by constructing the, possibly exponential sized, set of all maximal consistent subsets of a given set Γ and then attempting to construct a model directly by comparing when two such subsets can be related by the semantic binary relation. They are analogous to the construction of a canonical model when using a Henkin-style completeness proof for a Hilbert calculus for modal logic. If Γ is finite then so is its set of maximal consistent sets, so termination is usually easy [19], and it suffices to prove soundness and completeness constructively [11]. But they are not practical because their first task requires a possibly unnecessary exponential operation. More refined "on the fly" tableau procedures [29] only construct the set of subsets in the worst-case, and as is well-known, real-world examples rarely contain such worst-case examples.

Our work follows this "refined" approach. We break new ground for producing verified and optimized implementations for modal description logics by handling the basic modal logics K, KT and S4. For K, we also give a verified implementation of backjumping [3]. The logic K allows us to set the scene and incorporate backjumping. The logic KT allows us to showcase how to handle axiomatic extensions. The termination argument for the logic S4 requires detecting "ancestor loops" in tableau branches. Loop-checking is also required to handle knowledge bases (global assumptions), which encode real-world problems [4].

By utilizing the constructive nature and strong type system of Lean [9], we implemented verified decision procedures based on tableaux with histories, which are variants of sequent calculi with histories given by Heuerding, Seyfried, and Zimmermann [16] (HSZ henceforth). However, our formalization does not mimic the proofs given in HSZ. Although the verified decision procedures are not competitive against state-of-the-art provers, they provide promising evidence that efficient verified provers for expressive modal description logics are plausible.

The verified decision procedures are functions defined in Lean. They could be executed using `#reduce` but this will take too long to compute, meaning they cannot be used directly in a Lean proof. Instead, we use `#eval` to execute these functions on the virtual machine provided by Lean, and thus obtain our experimental results.

Related Work

Formalizations of theories, decision procedures and SAT solvers for classical propositional logic and first order logic have been well studied. Many recent verified provers also come with verified optimizations [7] [26]. These formalizations usually adopt modern variants of the resolution method as their primary calculus in order to achieve efficiency. On the other hand, there are also verified decision procedures based on tableau methods [21] [2] [17]. However, the target logics of these projects are either classical propositional logic [21] or basic description logics [2], without loop-checks, and without verified optimizations. For example, Hidalgo et al. [17] verified a decision procedure for description logic ALC, but their satisfiability was then defined with respect to empty global assumptions, so loop-checks are not required. Our formalization also extends their work. Other work related to modal and temporal logics includes Paulien [10], Bentzen [6] and Yuasa *et al.*[30]. Paulien [10] gives a comprehensive account of embedding modal logics in Coq. Bentzen [6] gives a formalization of Henkin-style completeness proof of modal logics in Lean. Yuasa et al. [30] use an external decision procedure for the μ-calculus to help verify the Deutsch-Schorr-Waite marking algorithm in Agda, but leave the decision procedure itself trusted. There are also formalizations of temporal logics developed by Schimpf et al. [25], Jantsch and Norrish [18], Esparza et al. [12], targeting verification related to model checking problems including verified model checkers, and translation between temporal logics and automata.

2 Modal logic preliminaries

Our verified decision procedures are all implemented with lists. However, for readability, we use usual mathematical notation for sets in the following when there is no confusion.

2.1 Syntax and semantics of K, KT and S4

▶ **Definition 2.1.** *The syntax of formulas in this paper is given by the following grammar.*

$$\mathbb{N} ::= 0 \mid S\ \mathbb{N}$$
$$\varphi ::= \mathbb{N} \mid \neg\mathbb{N} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \Box\varphi \mid \Diamond\varphi$$

▶ **Definition 2.2.** *The length l of a formula φ is the number of logical connectives including $\Box$ and $\Diamond$ occurring in φ. The length l of a set Γ of formulas is $\sum_{\varphi\in\Gamma} l(\varphi)$. The closure cl of a formula φ is the set of all the subformulas of φ.*

To obtain a neat formalization we only consider formulas in negation normal form as defined in Definition 2.1. However, it is easy to establish a translation between the full language and the negation normal form, preserving the correctness of the decision procedures.

▶ **Definition 2.3** (Kripke models). *A Kripke model is a triple (S, R, V) where S is a set of states, and $R \subseteq S \times S$ and $V \subseteq \mathbb{N} \times S$ are two binary relations. R is called a reachability relation, and V is called a valuation function.*

▶ **Definition 2.4.** *A KT model is a Kripke model whose reachability relation is reflexive. An S4 model is a KT model whose reachability relation is transitive.*

▶ **Definition 2.5** (forcing). *For a Kripke model $M = (S, R, V)$, the forcing relation $\Vdash$ between a state $s \in S$ and a formula φ is:*

$$
\begin{array}{ll}
(M,s) \Vdash n & \text{if } V(n,s) \\
(M,s) \Vdash \neg n & \text{if } (M,s) \nVdash n \\
(M,s) \Vdash \varphi \wedge \psi & \text{if } (M,s) \Vdash \varphi \text{ and } (M,s) \Vdash \psi \\
(M,s) \Vdash \varphi \vee \psi & \text{if } (M,s) \Vdash \varphi \text{ or } (M,s) \Vdash \psi \\
(M,s) \Vdash \Box\varphi & \text{if for all } t \in S, R(s,t) \text{ implies } (M,t) \Vdash \varphi \\
(M,s) \Vdash \Diamond\varphi & \text{if there exists } t \in S, R(s,t) \text{ and } (M,t) \Vdash \varphi
\end{array}
$$

▶ **Definition 2.6** (satisfiability). *Let M be a Kripke model. A state $s \in M$ satisfies a set Γ of formulas, written $(M, s) \vDash \Gamma$, if for all $\varphi \in \Gamma$, $(M, s) \Vdash \varphi$. A set Γ of formulas is satisfiable if there is a Kripke model containing a state that satisfies Γ, and is unsatisfiable otherwise.*

We write $s \Vdash \varphi$ and $s \vDash \Gamma$ if the model M is clear from the context. Kripke models are formalized as a Lean structure equipped with two relations, parameterized by a carrier type `states`. Also note that by definition, an empty model never satisfies a set Γ of formulas. When Γ is proved to be unsatisfiable, then it is also not satisfied in any non-empty model.

```
structure kripke (states : Type) :=
(val : ℕ → states → Prop)
(rel : states → states → Prop)

def sat {st} (k : kripke st) (s) (Γ : list nnf) : Prop :=
∀ φ ∈ Γ, force k s φ

def unsatisfiable (Γ : list nnf) : Prop :=
∀ (st) (k : kripke st) s, ¬ sat k s Γ
```

2.2 Tableaux for K, KT and S4

The tableau $\mathrm{K}^{\mathcal{T}}$ for modal logic K is the calculus defined as in Figure 1. We call the upper part of a rule the upper sequent (lower sequent resp.). Rule (K) is called the transition rule [13]. The computational behaviour of the transition rule has a backtracking flavor. If the lower sequent is unsatisfiable, then so is the upper sequent. If the lower sequent is satisfiable,

$$(id)\ \frac{n, \neg n, \Gamma}{\text{unsatisfiable}} \qquad (\wedge)\ \frac{\varphi \wedge \psi, \Gamma}{\varphi, \psi, \Gamma} \qquad (\vee)\ \frac{\varphi \vee \psi, \Gamma}{\varphi, \Gamma \quad \psi, \Gamma} \qquad (K)\ \frac{\Diamond\varphi, \Box\Sigma, \Gamma}{\varphi, \Sigma}$$

Figure 1 Tableau $\mathrm{K}^{\mathcal{T}}$.

$$(K)\ \frac{\Diamond\Delta, \Box\Sigma, \Gamma}{\varphi_0, \Sigma \quad \varphi_1, \Sigma \quad \ldots \quad \varphi_n, \Sigma}$$

where $\Delta = \{\varphi_0, \ldots, \varphi_n\} \neq \varnothing$ and Γ is a set of literals not containing a pair $n, \neg n$

Figure 2 Equivalent form of the transition rule.

then the decision procedure backtracks and tries another $\Diamond$-formula. If all of them are satisfiable, then so is the upper sequent. In our formalization, the transition rule should be understood as its variant as shown in Figure 2, which encodes such a computational behavior in the form of a rule. This applies to all the transition rules of the tableaux given in this paper. Rules with the semantics captured by our dotted line are also known as AND-nodes, indicating that such rules have a semantic interpretation that is "dual" to that of the $\vee$ rule, and the resulting calculi are also called AND-OR tableaux [14]. We abuse notation by using the same rule names but distinguish them by dotted lines.

▶ **Theorem 2.7** (invertibility). *For the (K) rule (above) and the $(\wedge)$ rule, all the lower sequents are satisfiable if and only if the upper sequent is satisfiable. One of the lower sequents of the $(\vee)$ rule is satisfiable if and only if the upper sequent is satisfiable.*

▶ **Theorem 2.8** (termination). *Let Γ_u be the upper sequent and Γ_l a lower sequent of a non-id rule in $K^{\mathcal{T}}$. Then $l(\Gamma_l) < l(\Gamma_u)$.*

The tableau $\mathrm{KT}^{\mathcal{T}}$ for modal logic KT is obtained by adding the (T) rule to $\mathrm{K}^{\mathcal{T}}$. The tableau $\mathrm{S4}^{\mathcal{T}}$ for modal logic S4 is $\mathrm{KT}^{\mathcal{T}}$ with the transition rule replaced by the rule $(S4)$:

$$(T)\ \frac{\Box\varphi, \Gamma}{\varphi, \Box\varphi, \Gamma} \qquad (S4)\ \frac{\Diamond\varphi, \Box\Sigma, \Gamma}{\varphi, \Box\Sigma}$$

3 Formalization

We now describe verified decision procedures for K, KT and S4. The one for K introduces the basic tools we developed for formalizing modal tableaux, and serves as an overview of the verified algorithms. The one for KT introduces tableaux with histories to handle non-trivial termination, and focuses on their correctness. The one for S4 combines the techniques for K and KT to deal with the correctness of loop-checks.

Each decision procedure is implemented as a computable function in Lean, and is proved to be sound, complete and terminating. They can be evaluated by Lean's `#eval` command.

3.1 Formalizing modal tableaux – K

The invertibility Theorem 2.7 guarantees that each rule of $\mathrm{K}^{\mathcal{T}}$ properly propagates the status of a sequent. Thus a natural way to write a decision procedure f for K is to call f recursively on the lower sequents of each rule and propagate the status upwards [29]. Eventually, the root sequent, which is the goal, will have its status updated. In a theorem prover supporting a strong type theory, the status can be a complex witness such as a proof or a model.

In cases where a formula φ is satisfiable, instead of returning a proof of the statement that φ is satisfiable, which is an existential sentence, we can return a Kripke model as a concrete object that satisfies φ. For the purpose of formalization, such a model should be easy to construct and easy to check. Defining a Kripke structure by specifying all its fields from scratch can be tedious, especially when one wants to extract information from a sequence of Kripke models and re-arrange them by manipulating their R and V to construct a new model. This happens when dealing with the transition rule. To achieve a better solution, we describe a uniform way of constructing Kripke models using tree models with interpretation functions, and let the decision procedure return such a model as a witness when a lower sequent is satisfiable.

A tree model is defined as an inductive type `model` with a first argument of type `list nat`, intuitively representing the propositional variables true in a state, and a recursive argument `list model`, intuitively representing the states reachable from that state. The base case is when the second list is empty and is not encoded explicitly. Interpretation functions `mval` and `mrel` are defined as follows to capture this intuition.

```
inductive model
| cons : list ℕ → list model → model

def mval : ℕ → model → bool
| p (cons v r) := p ∈ v

def mrel : model → model → bool
| (cons v r) m := m ∈ r
```

Note that although such a type is called model, as can be seen from the type of interpretation functions, `model` is supposed to be used as the type of a state. However, such a state contains essentially all the information about a Kripke model constructed so far. It is always possible to recover the model from within a state via the interpretation functions. We define such a recovery `builder`, whose type is exactly just `Kripke model`.

```
def builder : kripke model :=
{val := λ n s, mval n s, rel := λ s₁ s₂, mrel s₁ s₂} -- λ for coercion
```

This mechanism allows us to construct without too much effort a provably correct model of the upper sequent Γ of a transition rule from the models returned by its lower sequents. For example, if l is the list of tree models returned by the recursive calls on the lower sequents of the transition rule, then the tree model s of the upper sequent is simply $s := cons\ v\ l$ where v is a list of natural numbers definable from the upper sequent itself. For non-transition rules, the tree model remains the same. Then we prove `sat builder s` Γ. The return type of the decision procedure f is as follows, where the `//` notation denotes subtypes:

```
inductive node (Γ : list nnf) : Type
| closed : unsatisfiable Γ → node
| open_  : {s // sat builder s Γ} → node
```

Since the return value of calling f on the lower sequents of the transition rule is potentially a list of tree models satisfying each lower sequent, a predicate called `batch_sat` is defined to relate the lower sequents and their models. The function `unmodal` takes a sequent Γ and produces its lower sequent according to the transition rule.

```
inductive batch_sat : list model → list (list nnf) → Prop
| bs_nil : batch_sat [] []
| bs_cons (m Γ l₁ l₂) : sat builder m Γ → batch_sat l₁ l₂ →
                        batch_sat (m::l₁) (Γ::l₂)

-- unbox and undia take a list of formulas, and
-- get rid of the outermost box or diamond of each formula respectively
def unmodal (Γ : list nnf) : list (list nnf) :=
list.map (λ d, d :: (unbox Γ)) (undia Γ)
```

The verification of the transition rule is illustrated next. The type `modal_applicable` expresses the extra conditions of the transition rule: Γ contains only literals, contains no contradictions, and contains at least one $\Diamond$-formula. Moreover, `unmodal_sat_of_sat` not only encodes that if the upper sequent is satisfiable then so is every lower sequent, but also that it holds for any list Δ of formulas that contains all the $\Box$- and $\Diamond$-formulas in Γ.

```
theorem sat_of_batch_sat : Π l Γ (h : modal_applicable Γ),
batch_sat l (unmodal Γ) → sat builder (cons h.v l) Γ

theorem unmodal_sat_of_sat (Γ : list nnf) : ∀ (i : list nnf),
i ∈ unmodal Γ → (∀ {st : Type} (k : kripke st) s Δ
(h₁ : ∀ φ, box φ ∈ Γ → box φ ∈ Δ)
(h₂ : ∀ φ, dia φ ∈ Γ → dia φ ∈ Δ),  sat k s Δ → ∃ s', sat k s' i)
```

The termination of the algorithm is not difficult to formalize, because each lower sequent of each rule contains fewer logical connectives. A termination argument is then given by the length of a sequent. However, the transition rule requires some implementational attention. It is worth noting that a map-like function is needed to execute f on the list of lower sequents. Given a term such as `list.map f l` occurring within the definition of f, Lean does not know automatically that the computation terminates because f is not applied to any arguments. Secondly, the transition rule needs early termination in order to make the algorithm efficient. As soon as one of the lower sequents turns out to be unsatisfiable, the computation should terminate because the upper sequent must be unsatisfiable. We define a dedicated function as follows to achieve early termination and help Lean prove termination.

```
-- psum is the sum type extended to Sort in Lean.
def tmap {p : list nnf → Prop} (f : Π Γ, p Γ → node Γ):
Π Γ : list (list nnf), (∀ i∈Γ, p i) →
psum {i // i∈Γ ∧ unsatisfiable i} {x // batch_sat x Γ}
```

The dependent function f in the argument is an abstraction of the decision procedure f itself with a proof h saying that the input of it satisfies the property p. This p is supposed to be the termination of the transition rule whose proof is given as follows. Since the termination proof is found in the local context where the decision procedure f is being called, Lean knows that this recursive call terminates.

```
def unmodal_size (Γ : list nnf) : ∀ (i : list nnf),
i ∈ unmodal Γ → (node_size i < node_size Γ)
```

Since the return type contains either a proof that the goal is unsatisfiable, or a Kripke model which provably satisfies the goal, soundness and completeness are immediately given. They can also be proved explicitly as follows.

```
def tableau : Π Γ : list nnf, node Γ := ...
using_well_founded {rel_tac := λ _ _, `[exact ⟨_, measure_wf node_size⟩]}

def is_sat (Γ : list nnf) : bool :=
match tableau Γ with
| closed _ := ff
| open_ _  := tt
end

theorem correctness (Γ : list nnf) :
is_sat Γ = tt ↔ ∃ (st : Type) (k : kripke st) s, sat k s Γ
```

3.2 Tableaux with histories – KT

As we can see from the (T) rule of $\mathrm{KT}^{\mathcal{T}}$, the termination of proof search in KT becomes non-trivial. In HSZ, a sequent calculus with histories was proposed to handle termination. Soundness and completeness of such a sequent calculus was then proved by establishing a translation between the original calculus and the one with histories. We use a tableau system with histories based on the sequent calculus with histories, and give a direct semantic proof of soundness and completeness with the corresponding formalization. We use a different termination argument, as we found that the measure given by HSZ does not always decrease.

▶ **Definition 3.1.** *Tableau $KT^{\mathcal{TH}}$ is defined as in Figure 3 where the vertical bar | separates the history Σ, which is a formula-set, from the formula-set Γ carried by each sequent:*

$$(id)\ \frac{\Sigma \mid n, \neg n, \Gamma}{\text{unsatisfiable}} \qquad (\wedge)\ \frac{\Sigma \mid \varphi \wedge \psi, \Gamma}{\Sigma \mid \varphi, \psi, \Gamma} \qquad (\vee)\ \frac{\Sigma \mid \varphi \vee \psi, \Gamma}{\Sigma \mid \varphi, \Gamma \quad \Sigma \mid \psi, \Gamma}$$

$$(T)\ \frac{\Sigma \mid \Box\varphi, \Gamma}{\Box\varphi, \Sigma \mid \varphi, \Gamma} \qquad (K)\ \frac{\Box\Sigma \mid \Diamond\varphi, \Gamma}{\varnothing \mid \varphi, \Sigma}$$

where Γ in (K) is a set of literals not containing a contradiction

Figure 3 Tableau $\mathrm{KT}^{\mathcal{TH}}$.

▶ **Definition 3.2.** *A sequent $\Sigma \mid \Gamma$ is satisfiable if $\Sigma \cup \Gamma$ is satisfiable.*

The procedure to decide whether Γ is satisfiable in KT is similar to the one designed for K. Starting with the goal $\varnothing \mid \Gamma$ as a root sequent, apply rules repeatedly until a contradiction is found or no rule is applicable, whence a KT proof or model can be constructed.

However, the correctness proof now becomes different. The first thing to notice is that $\mathrm{KT}^{\mathcal{TH}}$ does not have the strict subformula property as does $\mathrm{K}^{\mathcal{T}}$. The lower sequent of the (T) rule contains more logical connectives than the upper sequent. Consequently, the termination argument that worked for K does not work for KT. Secondly, although the transition rule in $\mathrm{KT}^{\mathcal{TH}}$ is very similar to that of $\mathrm{K}^{\mathcal{T}}$, the change of semantics from K to KT means its invertibility is not immediately obvious. We prove termination by defining a measure on sequents and showing that such a measure decreases under a well-founded relation every time we apply a rule.

▶ **Definition 3.3.** *Let* Γ *be a set of formulas. The degree of* Γ *is the maximal number of modal operators occurring in any formula* $\varphi \in \Gamma$.

▶ **Definition 3.4.** *Let* $\Sigma \mid \Gamma$ *be a sequent in* $KT^{\mathcal{TH}}$. *The size of* $\Sigma \mid \Gamma$ *is defined as a pair*

$$size(\Sigma \mid \Gamma) := (degree(\Sigma \cup \Gamma), l(\Gamma))$$

▶ **Theorem 3.5.** *Let* $\Sigma \mid \Gamma$ *be the upper sequent and* $\Sigma' \mid \Gamma'$ *a lower sequent of a rule in* $KT^{\mathcal{TH}}$. *Let* $<_{lex}$ *be the lexicographic order on* $\mathbb{N} \times \mathbb{N}$. *Then*

$$size(\Sigma' \mid \Gamma') <_{lex} size(\Sigma \mid \Gamma)$$

Proof. For the (T) rule, $degree(\Sigma \mid \Gamma)$ remains unchanged and $l(\Gamma)$ decreases. For the transition rule, $degree(\Sigma \mid \Gamma)$ decreases. For propositional rules, there are two possibilities: either $degree(\Sigma \mid \Gamma)$ decreases or $degree(\Sigma \mid \Gamma)$ remains unchanged and $l(\Gamma)$ decreases. In either case $size(\Sigma \mid \Gamma)$ decreases. ◀

The invertibility of the transition rule is key to the correctness of the decision procedure. We prove this by establishing a semantic relationship between Σ and Γ of a sequent, using the tree model and interpretation functions mechanism developed in the previous section.

▶ **Definition 3.6.** *A sequent* $\Sigma \mid \Gamma$ *is called reflexive if for every* $\Box\varphi \in \Sigma$, *if a tree model* $m := cons\ v\ l$ *satisfies the following two conditions:*

1. $m \vDash \Gamma$, *and*
2. *for every* $s \in l$, *for every* $\Box\psi \in \Sigma$, $s \Vdash \psi$.

then $m \Vdash \varphi$.

▶ **Theorem 3.7.** *Let* $\Sigma \mid \Gamma$ *be a sequent generated by* $KT^{\mathcal{TH}}$. *Then*

1. Σ *contains only* $\Box$*-formulas.*
2. $\Sigma \mid \Gamma$ *is reflexive.*

A paper proof of Theorem 3.7 could proceed by induction on the construction of sequents. In our formalization, we encode Theorem 3.7 as a property into the definition of a sequent so a sequent cannot be constructed without proving it obeys Theorem 3.7. This avoids the extra work of defining an inductive type representing $\mathrm{KT}^{\mathcal{TH}}$, carrying out an explicit induction and relating such a type to the decision procedure. The final definition of a sequent of $\mathrm{KT}^{\mathcal{TH}}$ is:

```
structure seqt : Type :=
(main : list nnf)
(hdld : list nnf)
-- srefl main hdld says that sequent hdld | main satisfies theorem 3.7(2)
(pmain : srefl main hdld)
-- box_only says there are only boxed formulas in hdld
(phdld : box_only hdld)
```

As an example, we show the construction of a sequent in the implementation. The `and_child` function takes a sequent Γ, assumes that an $\wedge$-formula is found in the main part, and returns a new sequent which is the lower sequent of the $\wedge$-rule with Theorem 3.7 proved. Henceforth, we refer to this way of proving properties of sequents as "downward propagation". The function Γ`.main` returns the `main` field of the sequent Γ.

```
def and_child {φ ψ} (Γ : seqt) (h : nnf.and φ ψ ∈ Γ.main) : seqt :=
⟨φ :: ψ :: Γ.main.erase (and φ ψ), Γ.hdld,
 begin
   intros k s γ hsat hin hall,
   by_cases heq : γ = and φ ψ,
   { rw heq, split, apply hsat, simp, apply hsat, simp },
   { apply Γ.pmain _ hin hall,
     apply sat_and_of_sat_split _ _ _ _ _ h hsat }
 end, Γ.phdld⟩
```

▶ **Theorem 3.8** (invertibility). *All the lower sequents of the transition rule are satisfiable if and only if the upper sequent is satisfiable.*

Proof. ($\Rightarrow$) By Theorem 3.7. ($\Leftarrow$) By KT semantics. ◀

Note that applying Theorem 3.7 by itself gives us satisfiability with respect to tree models. However, in the end of the formalization, an immediate corollary is that a sequent is satisfiable if and only if it is satisfied by a tree model. Thus Theorem 3.8 does not claim too much.

3.3 Loop checks – S4

The transitivity constraint in the semantics of S4 poses more difficulties on both the termination and correctness of the decision procedure. HSZ introduced a sequent calculus with histories for S4 and proved its soundness and completeness via a translation connecting two intermediate calculi. We give a tableau calculus that enhances HSZ's calculus, and give a formalization of soundness and completeness without establishing translations. We again use a slightly different termination argument as the measure from HSZ could become negative in some cases. This does not necessarily mean that HSZ's argument is incorrect, because a negative lower bound might still be given. However, we were not able to find it in the paper.

▶ **Definition 3.9.** *Tableau $S4^{\mathcal{TH}}$ is defined as in Figure 4. The condition $\varphi \notin H$ in the transition rule is called a "loop-check". H, S and A are called a history, a signature, and ancestors of the sequent respectively. A signature is a pair of formula and list of formulas, but can be empty. The ancestors is a list of non-empty signatures. For each rule that is not id or S4, the χ in its upper sequent $A \,||\, S \,||\, H \,||\, \Sigma \mid \chi, \Gamma$ is called a principal formula.*

▶ **Definition 3.10.** *A sequent $A \,||\, S \,||\, H \,||\, \Sigma \mid \Gamma$ is satisfiable if $\Sigma \cup \Gamma$ is satisfiable. Given a sequent s, we refer to its fields by the projection notation (e.g., $s.\Gamma$).*

3.3.1 Downward propagation

The transition rule of $\mathrm{S4}^{\mathcal{TH}}$ prevents us from using the termination argument for $\mathrm{KT}^{\mathcal{TH}}$, because the degree of $\Sigma \mid \Gamma$ can remain unchanged, and the length of Γ can increase. However, $\mathrm{S4}^{\mathcal{TH}}$ has some nice properties that are helpful to design a termination argument.

▶ **Theorem 3.11.** *Let $A \,||\, S \,||\, H \,||\, \Sigma \mid \Gamma$ be a sequent generated by $S4^{\mathcal{TH}}$ from root $A' \,||\, S' \,||\, H' \,||\, \Sigma' \mid \Gamma'$. Then*

1. *Σ contains no duplicate elements. H contains no duplicate elements.*
2. *Σ and H are sublist permutations of $cl(\Gamma')$.*
3. *$\Gamma \subseteq cl(\Gamma')$.*

$$(id)\ \frac{A \mid\mid S \mid\mid H \mid\mid \Sigma \mid n, \neg n, \Gamma}{\text{unsatisfiable}}$$

$$(\wedge)\ \frac{A \mid\mid S \mid\mid H \mid\mid \Sigma \mid \varphi \wedge \psi, \Gamma}{A \mid\mid \varepsilon \mid\mid H \mid\mid \Sigma \mid \varphi, \psi, \Gamma} \qquad (\vee)\ \frac{A \mid\mid S \mid\mid H \mid\mid \Sigma \mid \varphi \vee \psi, \Gamma}{A \mid\mid \varepsilon \mid\mid H \mid\mid \Sigma \mid \varphi, \Gamma \quad A \mid\mid \varepsilon \mid\mid H \mid\mid \Sigma \mid \psi, \Gamma}$$

$$(\Box, \text{new})\ \frac{A \mid\mid S \mid\mid H \mid\mid \Sigma \mid \Box\varphi, \Gamma}{A \mid\mid \varepsilon \mid\mid \varnothing \mid\mid \Box\varphi, \Sigma \mid \varphi, \Gamma}\ (\Box\varphi \notin \Sigma)$$

$$(\Box, \text{dup})\ \frac{A \mid\mid S \mid\mid H \mid\mid \Sigma \mid \Box\varphi, \Gamma}{A \mid\mid \varepsilon \mid\mid H \mid\mid \Sigma \mid \varphi, \Gamma}\ (\Box\varphi \in \Sigma)$$

$$(S4)\ \frac{A \mid\mid S \mid\mid H \mid\mid \Sigma \mid \Diamond\varphi, \Gamma}{(\varphi, \Sigma), A \mid\mid (\varphi, \Sigma) \mid\mid \varphi, H \mid\mid \Sigma \mid \varphi, \Sigma}\ (\varphi \notin H)$$

where Γ in (S4) is a set of literals not containing a contradictory pair

Figure 4 Tableau $\mathrm{S4}^{\mathcal{TH}}$.

A paper proof of Theorem 3.11 could use induction on the $\mathrm{S4}^{\mathcal{TH}}$ rules using these three properties simultaneously. In the formalization, this is handled by a downward propagation as in Section 3.2.

▶ **Definition 3.12.** *Let $A \mid\mid S \mid\mid H \mid\mid \Sigma \mid \Gamma$ be a sequent generated by $S4^{\mathcal{TH}}$ from root $A' \mid\mid S' \mid\mid H' \mid\mid \Sigma' \mid \Gamma'$. We use $\circ$ for function composition, l for the length function and cl for the closure function in Definition 2.2. The size of $A \mid\mid S \mid\mid H \mid\mid \Sigma \mid \Gamma$ is a triple*

$$size(A \mid\mid S \mid\mid H \mid\mid \Sigma \mid \Gamma) := (l \circ cl(\Gamma') - l(\Sigma), l \circ cl(\Gamma') - l(H), l(\Gamma))$$

By Theorem 3.11, *size* is a well-defined function from sequents to $\mathbb{N} \times \mathbb{N} \times \mathbb{N}$.

▶ **Theorem 3.13.** *Let u be the upper sequent and l the lower sequent of a non-id rule in $S4^{\mathcal{TH}}$. Let $<_{lex}$ be the lexicographic order on $\mathbb{N} \times \mathbb{N} \times \mathbb{N}$. Then*

$$size(l) <_{lex} size(u)$$

In addition to Theorem 3.13, which suffices to prove termination, $\mathrm{S4}^{\mathcal{TH}}$ has more properties that help prove soundness and completeness. These properties do not need to be proved by referring to each other, but we gather them together as they are all properties about sequents, which can be handled by a downward propagation.

▶ **Theorem 3.14.** *Let $A \mid\mid S \mid\mid H \mid\mid \Sigma \mid \Gamma$ be a sequent generated by $S4^{\mathcal{TH}}$. Then*

1. *Σ contains only $\Box$-formulas.*
2. *For every $\varphi \in H$, $(\varphi, \Sigma) \in A$.*
3. *If $S \neq \varepsilon$ then $fst(S) \in \Gamma$ and $snd(S) \subseteq \Gamma$.*

As for a sequent of $\mathrm{KT}^{\mathcal{TH}}$, a sequent of $\mathrm{S4}^{\mathcal{TH}}$ can now be defined formally as follows.

```
structure sseqt : Type :=
(goal : list nnf)
(a : list psig) -- psig is the signature, which is of form (d, b)
(s : sig) -- sig := option psig
(h b m: list nnf)
(ndh : list.nodup h) -- nodup says there is no duplicate elements
```

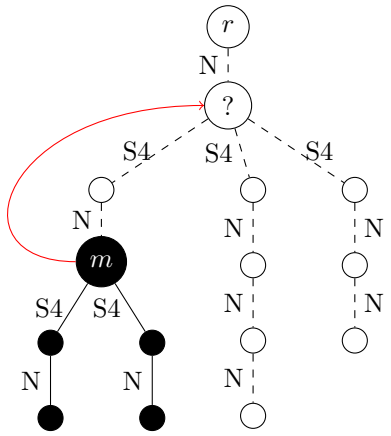

Figure 5 Edges labeled with S4 are an application of transition rule, N an application of non-transition rule. The red edge indicates that a loop-check is triggered at node m and a request is made. Black nodes are nodes with tree models constructed, and white nodes do not have a tree structure yet and their statuses are unknown to m. The node labeled r is the root.

```
(ndb : list.nodup b)
(sph : h <+~ closure goal) -- <+~ denotes sublist permutation
(spb : b <+~ closure goal)
(sbm : m ⊆ closure goal)
(ha : ∀ φ ∈ h, (⟨φ, b⟩ : psig) ∈ a)
(hb : box_only b)
-- dsig takes a signature (d, b) and a proof h, and returns d
(ps₁ : Π (h : s ≠ none), dsig s h ∈ m)
-- bsig takes a signature (d, b) and a proof h, and returns b
(ps₂ : Π (h : s ≠ none), bsig s h ⊆ m)
```

3.3.2 Upward propagation

Before diving into correctness, we give an informal view of the problems caused by S4. One essential difference between $\mathrm{S4}^{\mathcal{TH}}$ and $\mathrm{KT}^{\mathcal{TH}}$ is that when there are no rules applicable to a sequent l, in $\mathrm{KT}^{\mathcal{TH}}$ a provably correct model for l can immediately be constructed, but in $\mathrm{S4}^{\mathcal{TH}}$ this is not true. In $\mathrm{S4}^{\mathcal{TH}}$, there are two cases where no rules are applicable. The first case is that Γ contains only literals in the current sequent $A \mid\mid S \mid\mid H \mid\mid \Sigma \mid \Gamma$. In this case a tree model m which is a singleton can be constructed, and with some effort we might be able to prove $m \vDash \Sigma \cup \Gamma$.

The second case is that Γ contains not only literals, but also a list of $\Diamond$-formulas $\Diamond D$ such that $D \subseteq H$. This happens when loop-checks are triggered to prevent further computation. The intuition behind this termination is that if the $\Diamond$-formula to be handled occurs in the history, then it must have been handled before. Then a reachability relation is supposed to be established between the potential state that satisfies the current sequent and the potential state that satisfies the resulting sequent of the previous $(S4)$-rule application.

There are three levels of difficulty towards the construction of a provably correct model at this stage. The first is that the current sequent l needs to know where the previous handling happened and what the resulting sequent r was. The second is that even if it knows what r was, a tree model m_l for l cannot be constructed because r is above l and does not

have a tree structure yet. Moreover, m_l is a subtree of m_r and its construction should not refer to that of m_r. The third is that even if we give up the idea of trees and manage to construct a model where a state s_l is supposed to satisfy l (s_r for r resp.), to prove that s_l satisfies l, we need to show that all the $\Box$-formulas in l, when unboxed, are satisfied by s_r. However, whether this is true has not been determined because there could be unexplored branches of r. It is also worth noting that the overall status of r depends on the status of l, which is being determined. This non-well-founded behaviour of S4 is illustrated by Figure 5. Similar difficulties also occur when dealing with other more expressive modal logics such as propositional dynamic logic [29].

We proceed as follows to handle this non-well-foundedness and give a formalization of the correctness of $\mathrm{S4}^{\mathcal{TH}}$.

1. When no rule is applicable to a sequent $l = A \mid\mid S \mid\mid H \mid\mid \Sigma \mid \Gamma$ and Γ contains diamonds, a tree model m is constructed. The tree model comes with three additional pieces of data: the sequent l called *id*, a list of formulas called *htk*, and a list of signatures called *request*. Intuitively, *htk* contains formulas true within m, and *request* contains backward edges representing loops. A *request* can be defined using only H and Σ, without referring to a sequent occurring "above" l, but l does not know whether theses requests can be fulfilled.
2. The model m is then propagated to the upper sequents in the same way it is done for K and KT. For the transition rule, the constructor is applied to obtain a new tree. For the non-transition rules, the tree structure remains the same. The construction of *htk* and *request* are described below.
3. The correctness of m is left open at the time it is constructed, instead, a set P of properties of m is proved. These properties exploit the data contained in l and m, and are preserved by upward propagation. In other words, for each rule of $\mathrm{S4}^{\mathcal{TH}}$, if there is a tree model of the lower sequent with P proved, then a tree model of the upper sequent can also be constructed with P proved.
4. We show that if the root sequent $r = \varnothing \mid\mid \varepsilon \mid\mid \varnothing \mid\mid \varnothing \mid \Gamma$ has a tree model m_r with P proved, then interpretation functions can be defined on a type induced by m_r to construct an S4 model m. It can be proved from P that $m \vDash \Gamma$.

▶ **Definition 3.15.** *Lists htk and request are defined recursively as follows :*

1. *If no rules can be applied to a sequent $A \mid\mid S \mid\mid H \mid\mid \Sigma \mid \Gamma$, or it is the upper sequent of a transition rule, then*

   $$htk = \Gamma$$

   $$request = \{(\varphi, \Sigma) : \varphi \in H\}$$

2. *If htk_l and $request_l$ are defined for the a sequent of a non-transition rule R, then*

   $$htk_u = \{\varphi\} \cup htk_l$$

   $$request_u = request_l$$

   where φ is the principal formula of R.

▶ **Definition 3.16.** *A list l of formulas is called pre-hintikka if the following hold:*

1. *For every propositional variable p, if $p \in l$ then $\neg p \notin l$.*
2. *If $\varphi \wedge \psi \in l$, then $\varphi \in l$ and $\psi \in l$.*
3. *If $\varphi \vee \psi \in l$, then $\varphi \in l$ or $\psi \in l$.*
4. *If $\Box\varphi \in l$, then $\varphi \in l$.*

▶ **Theorem 3.17.** *Let m be a tree model and $A \parallel S \parallel H \parallel \Sigma \mid \Gamma$ its id. Then $m.htk$ is pre-hintikka and $\Gamma \subseteq m.htk$.*

As for sequents, Theorem 3.17 can be part of the definition of a tree model. We put *id*, *htk*, and Theorem 3.17 into a single package called `info`, as they are the information about the state being constructed. The final definition of an S4 tree model is as follows:

```
structure info : Type :=
(id : sseqt)
(htk : list nnf)
(hhtk : pre_hintikka htk)
(mhtk : Γ.m ⊆ htk)

inductive tmodel
| cons : info → list tmodel → list psig → tmodel
```

In order to describe the properties P in step 3, we need one more definition.

▶ **Definition 3.18.** *Let $m := cons\ i\ l\ r$ be a tree model. A tree model s is a child of m if $s \in l$. The descendant relation is the transitive closure of the child relation.*

The following non-trivial properties are the key to the correctness proof. We refer to the *id*, *htk* and *request* of a tree model m by the projections $m.id$, $m.htk$ and $m.request$ respectively.

▶ **Theorem 3.19.** *Let m be a tree model constructed in the way described above. Then*

1. *If s is a child of m and $\varphi \in m.id.\Sigma$, then $\varphi \in s.htk$.*
2. *If s is a child of m and $\Box\varphi \in m.htk$, then $\Box\varphi \in s.htk$.*
3. *If $\Diamond\varphi \in m.htk$, then either there exists a Δ such that $(\varphi, \Delta) \in m.request$ or there exists a child s of m such that $\varphi \in s.htk$.*
4. *If $(\gamma, \Delta) \in m.request$ and $\Box\varphi \in m.htk \cup m.id.\Sigma$, then $\Box\varphi \in \Delta$.*
5. *$m.request \subseteq m.id.A$.*

Property 2 requires property 1 as a lemma, and property 5 needs property 2 from Theorem 3.14. We omit the proof of Theorem 3.19, but give a proof sketch of the following substantial theorem, which illustrates that well-founded reasoning is achieved eventually.

▶ **Theorem 3.20** (fulfillment). *Let m be a tree model constructed in the way described above and s be a descendant of m. For every $r \in s.request$, either $r \in m.id.A$, or there exists a descendant d of m such that $r = d.id.S$.*

Proof. By induction on the construction of m. In the base case, there is no descendant of m. If m is constructed by non-transition rules, the theorem holds trivially because the tree structure, $m.id.A$ and *request* remain the same. Suppose m is constructed by the transition rule. Let s be a descendant of m. Then s is either a child of m or there exists a child c of m such that s is a descendant of c. In the first case, we proceed by cases on whether r is the head of $s.id.A$: if so, then s itself is the witness of a qualified descendant, else $r \in m.id.A$. In the second case, we apply the inductive hypothesis and proceed by cases once more. ◀

The fulfilment theorem tells us that every request is eventually fulfilled by the tree model constructed at the root. This is because the root sequent has an empty *ancestors* A.

▶ **Theorem 3.21** (global invariant). *Let m be a tree model constructed in the way described above and s a descendant of m. s satisfies the conclusions of Theorem 3.19 and Theorem 3.20.*

Theorem 3.21 is not superfluous for the formalization. Since Theorem 3.19 and Theorem 3.20 are not part of the definition of a tree model, each model knows that it itself satisfies Theorem 3.19 and Theorem 3.20, but has no information about its descendant once the construction is completed. The formalization of Theorem 3.21 is no harder than a proof by intuition – it is simply a property of m whose proof is immediately given by Theorem 3.19 and Theorem 3.20 during the construction of m, and can be kept by the return type.

For convenience, we now define a subtype `rmodel` that combines a tree model and the invariants `ptmodel`, as the invariants are frequently referred to in the following proofs, especially in proving semantic facts about the reachability relation. The evaluation function `val` and reachability relation `reach` are defined on `rmodel` as follows. Note that `reach` is the reflexive transitive closure of the relation `reach_step`. A state s reaches a state t by one step, if t is a child of s, or the *signature* of t is in the *request* of s.

```
def rmodel : Type := {m : tmodel // ptmodel m}

inductive reach_step : rmodel → rmodel → Prop
| fwd (s : rmodel) (i l ba h) : s.1 ∈ l → reach_step ⟨(cons i l ba), h⟩  s
| bwd (s : rmodel) (i l ba h) : (∃ rq ∈ ba, some rq = msig s.1) →
                                reach_step ⟨(cons i l ba), h⟩ s

def reach (s₁ s₂ : rmodel) := rtc reach_step s₁ s₂
```

Given a tree model m, the carrier set M of the final S4 model induced by m is all the `rmodel`s whose `tmodel` part is either m or a descendant of m. The final S4 model ready to be proved correct is as follows.

```
def builder (m : tmodel) : S4 {x : rmodel // x.1 = m ∨ desc x.1 m} :=
{val := λ v s, var v ∈ htk s.1.1,
 rel := λ s₁ s₂, reach s₁ s₂, -- λ for coercion
 refl := λ s, refl_reach s,
 trans := λ a b c, trans_reach a b c}
```

When there is no confusion, we refer to the *htk* of an element s in M as $s.htk$.

▶ **Theorem 3.22.** *Let m be the tree model returned by the decision procedure called on the root sequent $r = \varnothing \,||\, \varepsilon \,||\, \varnothing \,||\, \varnothing \mid \Gamma$ with Theorem 3.19, Theorem 3.20 and Theorem 3.21 proved. Then for every state s in the induced model M, if $\varphi \in s.htk$ then $s \Vdash \varphi$. In particular, when viewed as an element of M, $m \vDash \Gamma$.*

Proof. By induction on the construction of formulas. This one makes use of everything proved so far, especially the invariants. $m \vDash \Gamma$ because $\Gamma \subseteq m.htk$ by Theorem 3.17. ◀

4 Backjumping

The verified decision procedures described above can be equipped with provably correct optimizations as well. We now describe how backjumping [3] as an optimization can be integrated to gain exponential speedups.

Backjumping reduces search space by preventing recursive calls on the right branch of an $(\vee)$ rule when its status can already be determined by analyzing the information propagated from the left branch. If the left branch is open, then there is no need for backjumping as we don't have to explore the right branch. Backjumping is triggered only on closed branches. On

the other hand, all the significant changes we made to verify $\mathrm{KT}^{\mathcal{TH}}$ and $\mathrm{S4}^{\mathcal{TH}}$ happen only on the construction of models, namely the open branches, and the closed branches remain almost the same. Due to this nature, the verification of backjumping does not interfere with the proofs in Section 3.2 and Section 3.3. Such a verification for K can be ported to KT and S4 without too many changes. We formalize backjumping for K as an example, and leave the extension to KT and S4 as future work.

We define a notion of responsibility for each of the rules of $\mathrm{K}^{\mathcal{T}}$. Each sequent is assigned a list of formulas, called a marking set, representing the formulas responsible for contradictions. The construction of a marking set is an upward propagation.

▶ **Definition 4.1** (responsibility). *A marking set M is recursively defined on closed branches as follows. The φ and ψ refer to their corresponding occurrences in $K^{\mathcal{T}}$ defined in Figure 1.*

1. *For the id rule, $M = \{p, \neg p\}$.*
2. *Let M_l be the marking set of the lower sequent of the $\wedge$-rule.*

$$M = \begin{cases} \{\varphi \wedge \psi\} \cup M_l & \text{if } \varphi \in M_l \text{ or } \psi \in M_l \\ M_l & \text{otherwise} \end{cases}$$

3. *Let M_l/M_r be the marking sets of the left/right lower sequent of the $\vee$-rule respectively.*

$$M = \begin{cases} M_l \cup M_r \cup \{\varphi \vee \psi\} & \text{if } \varphi \in M_l \text{ or } \psi \in M_r \\ M_l \cup M_r & \text{otherwise} \end{cases}$$

4. *Let l be the first unsatisfiable lower sequent of the transition rule, with a marking set M_l:*

$$M = \Diamond(l.head) \cup \Box(l.tail \cap M_l)$$

The idea of backjumping is that if the left principal formula (i.e., φ) of the ($\vee$) rule is not in the marking set of the left lower sequent, then the upper sequent is unsatisfiable. We strengthen this claim and prove the following:

▶ **Theorem 4.2** (marking). *For each sequent Γ, if a sublist Δ of Γ contains nothing in the marking set if defined, then $\Gamma - \Delta$ is unsatisfiable.*

Formally, Theorem 4.2 is defined as:

```
def pmark (Γ m : list nnf) :=
∀ Δ, (∀ δ ∈ Δ, δ ∉ m) → Δ <+ Γ → unsatisfiable (list.diff Γ Δ)
```

The motivation of Theorem 4.2 is that we want to add a new rule BJ standing for backjumping into $\mathrm{K}^{\mathcal{T}}$. The upper sequent is immediately unsatisfiable because Γ is unsatisfiable by Theorem 4.2. Also note that a marking set is defined if and only if the sequent is unsatisfiable.

$$(BJ)\ \frac{\varphi \vee \psi; \Gamma}{\varphi; \Gamma} \quad \text{if } \varphi \notin M_l$$

In terms of the formalization of S4, Theorem 4.2 is an invariant. It is proved during the upward propagation along with the construction of the marking set, but this time everything happens in the closed branches. The formalization of Theorem 4.2 is difficult, mainly due to the reasoning about list difference. We omit the proof here as it should be conceptually clear how it can be proved by induction. One thing to note is that a marking set of BJ also needs to be defined and proved to respect Theorem 4.2 because BJ now takes part in the computation. This can be achieved by using case 3 of Definition 4.1 assuming M_r is empty. The corresponding proof of Theorem 4.2 is then straightforward.

Table 1 Results on the LWB benchmark for K (left) and S4 (right).

Subclass	K	K (backjumping)	FaCT++
branch_n	3	5	10
branch_p	1	3	10
d4_n	5	5	21
d4_p	6	7	21
dum_n	18	18	21
dum_p	9	17	21
grz_n	21	21	21
grz_p	6	7	21
lin_n	3	4	21
lin_p	6	7	21
path_n	10	10	21
path_p	2	12	21
ph_n	3	3	13
ph_p	2	3	7
poly_n	20	20	21
poly_p	19	21	21
t4p_n	7	7	21
t4p_p	7	12	21

Subclass	S4	FaCT++
45_n	0	21
45_p	1	21
branch_n	3	6
branch_p	6	7
grz_n	21	21
grz_p	0	21
ipc_n	3	10
ipc_p	3	10
md_n	3	10
md_p	4	4
path_n	2	15
path_p	1	21
ph_n	3	7
ph_p	2	6
s5_n	2	21
s5_p	21	21
t4p_n	0	21
t4p_p	0	21

5 Evaluation

We evaluated the performance of the verified decision procedures for K and S4 against FaCT++ [28], using the Logic Work Bench (LWB) benchmarks [5]. FaCT++ is a state-of-the-art reasoner for modal description logics. The LWB benchmarks are widely used for measuring the performance of modal reasoners [20] [15]. There are 18 subclasses of problems in the benchmark. Each subclass contains 21 problems, and each problem is harder to solve than the previous ones within the same subclass. We perform the tests on an Intel 2.20 GHz CPU with 2GB of memory. The time limit for each problem in a subclass is set to be 100 seconds and Table 1 shows the most difficult problem solved from each subclass by each prover within this limit. Thus the first row of the left hand table shows that our verified provers K and K (with backjumping) could solve 3 and 5 problems, respectively, while FaCT++ could solve 10, with each problem taking at most 100 seconds.

It is not surprising that FaCT++ outperforms the verified decision procedures on almost every problem. Figure 6 displays a fragment of a typical profile of the verified S4 decision procedure called on a problem. It can be seen that nearly 50% to 75% of the time is spent on `dec_eq_nnf`, which is called heavily in list operations such as `list.erase`. This suggests that future improvements of efficiency can include using better data structures such as arrays or hash tables instead of lists, as well as implementing other algorithmic optimizations such as unit propagation, semantic branching [28] and better ordering heuristics [27]. On the other hand, we see from Table 1 that the decision procedure for K with backjumping dominates the vanilla one in performance and backjumping is never worse. In particular, on the subclasses `dum_p` and `path_p`, there is a huge boost given by backjumping.

6 Conclusion and future work

We have presented verified decision procedures for three basic modal logics and shown how to handle loop-checking and backjumping. All of these decision procedures are executable, and are proved to be sound, complete and terminating. Backjumping has been implemented and

```
#eval execution took 15.6s
    15552ms    99.9%   tableau
    ...
    10703ms    68.8%   dec_eq_nnf
     9491ms    61.0%   list.decidable_mem
    ...
```

Figure 6 A fragment of a typical profile.

verified for K, and can be ported to KT and S4. All of these decision procedures return a concrete Kripke model when the input set of formulas is satisfiable, and a proof constructed via the tableau rules witnessing unsatisfiability otherwise. In fact, although the following well-known theorem is not formalized, it is implied by the formalization:

▶ **Theorem 6.1** (finite model K, KT, S4)**.** *Every satisfiable formula is satisfiable in a finite model.*

It should be clear that each satisfiable formula is witnessed by a tree model, which is a finite object. It can also be seen that the valuation functions and reachability relations constructed by the decision procedures for K and KT are computable. In the case of S4, this is a bit subtle because the transitive closure relation is not necessarily computable. However, since the descendants of a tree model form a finite set, by checking their *request*s and *signature*s one by one, we do have a way to compute reachability. We leave it to future work to have an explicit formalization of this for completeness.

Tableaux with histories offer us convenient tools for formalizing correctness of decision procedure for modal logics, but they also introduce some inefficiency. Comparing to a sequent of $\mathrm{S4}^{\mathcal{T}}$, a sequent of $\mathrm{S4}^{\mathcal{TH}}$ contains more information and takes time to construct. The extra information helps with verification but slows down the implementation. Therefore, future work also includes finding a balance point between the ease of formalization and computational efficiency of these decision procedures, and of course, porting backjumping to them would be an external boost.

One last thing to notice is that the expressiveness of S4 allows us to apply the verified decision procedures to more than modal logics. Since S4 is topologically complete [22], a translation between Kripke semantics and topological semantics can be established. It can be shown that a formula φ has a topological model if and only if it has an S4 model. We have also done half of this translation in our formalization. Another translation called the Gödel-McKinsey-Tarski translation is given in McKinsey and Tarski [23]. It is a translation between propositional intuitionistic logic and modal logic S4, and preserves theoremhood. Consequently, if the translation is formalized, then a verified decision procedure for S4 also gives us a verified decision procedure for intuitionistic propositional logic. The formalization of S4 opens the possibility of promising cross-field applications, and we leave the implementation of these as future work.

References

1 Samson Abramsky. Domain Theory and the Logic of Observable Properties. *CoRR*, abs/1112.0347, 2011. `arXiv:1112.0347`.

2 José-Antonio Alonso, Joaquín Borrego-Díaz, María-José Hidalgo, Francisco-Jesus Martín-Mateos, and José-Luis Ruiz-Reina. A Formally Verified Prover for the ALC Description Logic. In *Proceedings of the 20th International Conference on Theorem Proving in Higher Order Logics*, TPHOLs'07, pages 135–150, Berlin, Heidelberg, 2007. Springer-Verlag. URL: `http://dl.acm.org/citation.cfm?id=1792233.1792244`.

3 Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, New York, NY, USA, 2003.

4 Franz Baader and Bernhard Hollunder. A terminological knowledge representation system with complete inference algorithms. In Harold Boley and Michael M. Richter, editors, *Processing Declarative Knowledge*, pages 67–86, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

5 Peter Balsiger, Alain Heuerding, and Stefan Schwendimann. A Benchmark Method for the Propositional Modal Logics K, KT, S4. *Journal of Automated Reasoning*, 24(3):297–317, April 2000. doi:10.1023/A:1006249507577.

6 Bruno Bentzen. A Henkin-style completeness proof for the modal logic S5, 2019. URL: https://github.com/bbentzen/mpl.

7 Jasmin Christian Blanchette, Mathias Fleury, Peter Lammich, and Christoph Weidenbach. A Verified SAT Solver Framework with Learn, Forget, Restart, and Incrementality. *Journal of Automated Reasoning*, 61(1):333–365, June 2018. doi:10.1007/s10817-018-9455-7.

8 Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999.

9 Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The Lean Theorem Prover (System Description). In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25*, pages 378–388, Cham, 2015. Springer International Publishing.

10 Paulien de Wind. Modal Logic in Coq. Master's thesis, Vrije Universiteit Amsterdam, 2001.

11 Christian Doczkal and Joachim Bard. Completeness and Decidability of Converse PDL in the Constructive Type Theory of Coq. In *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2018, pages 42–52, New York, NY, USA, 2018. ACM. doi:10.1145/3167088.

12 Javier Esparza, Peter Lammich, René Neumann, Tobias Nipkow, Alexander Schimpf, and Jan-Georg Smaus. A fully verified executable LTL model checker. In *International Conference on Computer Aided Verification*, pages 463–478. Springer, 2013.

13 Rajeev Goré. Tableau Methods for Modal and Temporal Logics. In Marcello D'Agostino, Dov M. Gabbay, Reiner Hähnle, and Joachim Posegga, editors, *Handbook of Tableau Methods*, pages 297–396. Springer Netherlands, Dordrecht, 1999. doi:10.1007/978-94-017-1754-0_6.

14 Rajeev Goré. AND-OR tableaux for fixpoint logics with converse: LTL, CTL, PDL and CPDL. In Stéphane Demri, Deepak Kapur, and Christoph Weidenbach, editors, *Automated Reasoning*, pages 26–45, Cham, 2014. Springer International Publishing.

15 Rajeev Goré, Kerry Olesen, and Jimmy Thomson. Implementing Tableau Calculi Using BDDs: BDDTab System Description. In Stéphane Demri, Deepak Kapur, and Christoph Weidenbach, editors, *Automated Reasoning*, pages 337–343, Cham, 2014. Springer International Publishing.

16 Alain Heuerding, Michael Seyfried, and Heinrich Zimmermann. Efficient loop-check for backward proof search in some non-classical propositional logics. In P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, editors, *Theorem Proving with Analytic Tableaux and Related Methods*, pages 210–225, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

17 M. J. Hidalgo-Doblado, J. A. Alonso-Jiménez, J. Borrego-Díaz, F. J. Martín-Mateos, and J. L. Ruiz-Reina. Formally Verified Tableau-Based Reasoners for a Description Logic. *Journal of Automated Reasoning*, 52(3):331–360, March 2014. doi:10.1007/s10817-013-9291-8.

18 Simon Jantsch and Michael Norrish. Verifying the LTL to Büchi Automata Translation via Very Weak Alternating Automata. In Jeremy Avigad and Assia Mahboubi, editors, *Interactive Theorem Proving*, pages 306–323, Cham, 2018. Springer International Publishing.

19 Mark Kaminski, Thomas Schneider, and Gert Smolka. Correctness and Worst-Case Optimality of Pratt-Style Decision Procedures for Modal and Hybrid Logics. In Kai Brünnler and George Metcalfe, editors, *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 196–210, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

20 Mark Kaminski and Tobias Tebbi. InKreSAT: Modal reasoning via incremental reduction to SAT. In Maria Paola Bonacina, editor, *CADE-24*, volume 7898 of *LNCS*, pages 436–442. Springer, June 2013.

21 LudvikGalois. A verfied tableau prover for classical propositional logic, 2018. URL: `https://github.com/LudvikGalois/coq-CPL-NNF-tableau`.

22 J. C. C. McKinsey and Alfred Tarski. The Algebra of Topology. *Annals of Mathematics*, 45(1):141–191, 1944. URL: `http://www.jstor.org/stable/1969080`.

23 John CC McKinsey and Alfred Tarski. Some theorems about the sentential calculi of Lewis and Heyting. *The journal of symbolic logic*, 13(1):1–15, 1948.

24 John-Jules Ch Meyer and Wiebe Van Der Hoek. *Epistemic Logic for AI and Computer Science*. Cambridge University Press, New York, NY, USA, 1995.

25 Alexander Schimpf, Stephan Merz, and Jan-Georg Smaus. Construction of Büchi Automata for LTL Model Checking Verified in Isabelle/HOL. In *TPHOLs*, 2009.

26 Anders Schlichtkrull. Formalization of the Resolution Calculus for First-Order Logic. *Journal of Automated Reasoning*, 61(1):455–484, June 2018. `doi:10.1007/s10817-017-9447-z`.

27 Dmitry Tsarkov and Ian Horrocks. Ordering Heuristics for Description Logic Reasoning. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, pages 609–614, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc. URL: `http://dl.acm.org/citation.cfm?id=1642293.1642391`.

28 Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In Ulrich Furbach and Natarajan Shankar, editors, *Automated Reasoning*, pages 292–297, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

29 Florian Widmann. *Tableaux-based Decision Procedures for Fixed Point Logics*. PhD thesis, Australian National University, 2010.

30 Yoshifumi Yuasa, Yoshinori Tanabe, Toshifusa Sekizawa, and Koichi Takahashi. Verification of the Deutsch-Schorr-Waite Marking Algorithm with Modal Logic. In *Proceedings of the 2Nd International Conference on Verified Software: Theories, Tools, Experiments*, VSTTE '08, pages 115–129, Berlin, Heidelberg, 2008. Springer-Verlag. `doi:10.1007/978-3-540-87873-5_12`.