Free-Variable Tableaux for Efficient Deduction in K

Jens Happe

School of Computing Science
Simon Fraser University
Burnaby, B.C., V5A 1S6, CANADA
jhappe@cs.sfu.ca

Abstract. Free variable-tableaux have proven to be an adequate calculus for propositional modal logics. However, they have not materialized into efficient theorem provers. In this paper, we present a variant of the free-variable calculus presented in [3]. The distinctive features of our formalism are the use of labels throughout the formulas (not only as prefixes), with anonymous placeholders for universal variables, and a rigid definition of a free-variable model. We give a proof for soundness and completeness of our calculus for the modal logic **K**, provide a decision procedure which returns a model for any satisfiable formula, and outline how standard simplification techniques can be used to optimize the algorithm.

1 Introduction

The modal logic **K**, and more so, the Description Logic counterpart of its multi-modal version, $\mathcal{ALC}$, have established themselves in the field of Automated Reasoning. Extensions of $\mathcal{ALC}$ have proven to be adequate formalisms for reasoning with knowledge and are now used as underlying formalisms of medical knowledge bases, verification problems, and more recently, one of the Semantic Web languages [16]. Highly optimized theorem provers [14, 15, 11] provide reasoning services for these knowledge bases and problems. Secondly, the complexity class of satisfiability checking in **K**, PSPACE, poses interesting challenges: while still fully decidable, it strictly exceeds the complexity of propositional logic. While the performance of systems in real-world applications is encouraging, it has been observed that on encodings of QBF problems which (in the unbounded case) attain the PSPACE complexity bound, the performance of modal and description logic theorem provers still lags far behind that of dedicated QBF solvers [20]. Also, existing provers do not meet the theoretically possible EXPTIME bound in running time for general $\mathcal{ALC}$ problems [6]. Among the suggested remedies are simplification rules [19], nogood caching [6], and structural subsumption tests [13]. In the prototype prover ModProf [12], it was demonstrated that extensive simplification can indeed lead to improved performance on benchmark problems.

It is known that even for PSPACE-complete problems, minimal satisfying Kripke models may be exponential in space. For example, Ladner encodings [18] of QBF problems tend to entail exponential-size minimal Kripke models. In the classical tableau approach, each world must be visited, which inevitably leads to exponential running time or worse[1]. (This observation is reflected in the results reported in [21], especially those on Ladner-encoded QBF formulas.)

Free-variable tableau calculi are common in first-order logics, and they have also been proposed for modal logics [1, 3, 10]. Essentially, all formulas are assigned *labels* which, unlike the labels in Fitting's labelled tableau calculus [7], may contain placeholders to allow reasoning in several worlds, concepts or instances at once. As a result, the search space may be greatly reduced compared to conventional tableaux [3]. [2]. Another appeal of free-variable tableaux is their flexibility with regard to the particular logic being modelled. With only slight variations in the formalism, all basic normal logics can be modelled [3]. Furthermore, a decision procedure for some logics, including **K**, has been provided and implemented in the theorem prover **LeanK** [2]. However, a lean implementation has proven very inefficient, and we know of no published system which competes in performance with the leaders in the field.

In this paper, we propose a free-variable representation format for formulas of the modal logic **K**[3], which preserves satisfiability. The novel features are that modal operators within a formula are entirely replaced by labels, which are composed of constants and occurrences of a *wildcard* $*$ which can be instantiated by any constant. This allows reasoning about several accessible worlds at once, and at any level of nesting within the formula. The semantics are defined in terms of *free-variable models*, which are sets of labelled literals whose labels may also contain the $*$ symbol, allowing a more compact representation than Kripke models do. The free-variable formalism in its entirety is introduced in Section 3 of this paper.

In Section 4, we will proceed to devise a tableau calculus for free-variable formulas, prove it sound and complete, and derive a decision procedure. In particular, we will show how a complete open branch provides a free-variable model. It turns out that, after suitable preprocessing of the original formula, the tableau calculus needs only one rule, in contrast to the four different types of rules in the traditional tableau calculus for **K**[7] or the free-variable calculus in [3].

Our calculus, though independently developed, can be seen as a variant of the calculus in [3]. In Section 5, we will compare the two approaches and highlight the differences, showing how our calculus can be derived as a modification of the latter. In the last section, we will give an outlook, with a specific focus on the usefulness of our algorithm in deriving an efficient theorem prover, possible simplification techniques, and implementation challenges.

[1] Model caching and model merging may alleviate the consequences but do not address the problem.

[2] Which is not to say that all satisfiable problems have polynomial-size free-variable models. In fact, formulas with at least exponential-size models can be readily constructed.

[3] The restriction to **K** was made chiefly for brevity. A generalization to $\mathcal{ALC}$ is straightforward, and we suggest that the formalism can be easily adapted to logics such as **KT** and **S4**, as well as extensions with global and local assumptions.

2 Basic Concepts

We define the language of propositional modal logic as the set of all well-formed expressions made from *atoms* (a set P of *propositional variables*, augmented by the propositional constants $\top$ and $\bot$), the unary operators $\neg$, $\Box$ and $\Diamond$ (the latter two are the *modal operators*), and the binary operators $\wedge$ and $\vee$. (The expression $x \rightarrow y$ is considered a variant of $\neg x \vee y$.) Atoms and negated variables are also called *positive* and *negative literals*, respectively. The *length* of a formula F (written $l(F)$) is defined as the total number of occurrences of atoms. Any substring of a formula F which is itself a well-formed formula is called a *subformula*. The set $\text{vars}(F)$ is defined as the set of variables mentioned anywhere in F. The *modal depth* of F, $d(F)$, is defined to be 0 for atoms, $1 + d(G)$ for any formula of the form $\Box G$ or $\Diamond G$, and the maximal modal depth of any proper subformula in F, for any other F. A formula is said to be in *Negation Normal Form* (NNF) if negations occur only in front of atoms.

A thorough treatment of the semantics of the modal logic **K** can be found in standard works such as [4]. Here we only define it by way of *Kripke models*, originally introduced in [17]. A *Kripke model* (W, R, V) consists of a finite, non-empty set W of *possible worlds*, an *accessibility relation* R defined on W, and a *valuation function* $V : W \times \mathcal{P} \mapsto \{\mathit{true}, \mathit{false}\}$. We further require the graph (W, R) to be a directed tree with root w_0[4]. Given a Kripke model, we recursively define the conditions under which a world $w \in W$ *satisfies* a formula F (written $w \models F$) or not ($w \not\models F$), depending on the main operator in F:

$w \models \top$.
$w \not\models \bot$.

$w \models x$	if $V(w, x) = \mathit{true}$
$w \models \neg F$	if $w \not\models F$
$w \models \alpha_1 \wedge \alpha_2$	if $w \models \alpha_1$ and $w \models \alpha_2$
$w \models \beta_1 \vee \beta_2$	if $w \models \beta_1$ or $w \models \beta_2$
$w \models \Box F$	if for all $w' \in W$ such that wRw', $w' \models F$.
$w \models \Diamond F$	if there exists some $w' \in W$ such that wRw' and $w' \models F$.

A Kripke model in which $w_0 \models F$ is called a *(satisfying) model of F*. We say that a formula is **K**-*valid* (*valid* for short) if all Kripke models satisfy F. Conversely, a formula is called *invalid* (*unsatisfiable*), whenever there is no Kripke model satisfying F, and *satisfiable* otherwise. We extend all these notions to sets of formulas, defining $w \models S$ if $w \models F$ for all F in the set S.

Obviously, a formula is invalid iff its negation is valid. Therefore, to prove a formula F valid, we can show that $\neg F$ has no model.

3 The Free-Variable Formalism

Let C be a fixed infinite set of *constants*, and $*$ a symbol not contained in C, the so-called *wildcard*.

[4] The restriction is justified since **K** satisfies the tree model property. Graphs with loops or converging branches are useful but will not be considered for lack of space.

A *label* of *length* n is an n-tuple of symbols from $C \cup \{*\}$. The kth element of a label is called its kth *position*. The *concatenation* of two labels σ_1, σ_2 is defined in the obvious way and denoted $\sigma_1\sigma_2$. The only label of length 0 is the *empty label*, denoted ε. We identify labels of length 1 with their one element, e.g. $\sigma = c$. Label σ_1 is called a *prefix* of σ, if there exists a label σ_2 (possibly empty) such that $\sigma = \sigma_1\sigma_2$. A label is called *ground*, if it does not contain any occurrence of $*$. Label σ_0 is called an *instance* of σ, if σ_0 and σ are of the same length, and in the only positions where σ and σ_0 differ, σ has a $*$. Two labels σ_1 and σ_2 *unify* if they have a common instance (*unifier*); the *most general unifier* (mgu) (σ_1, σ_2) is the unifier with the maximum number of $*$ positions.

3.1 Labelled Formulas

Definition 1. *A* labelled formula *is inductively defined as follows:*

- *All (unlabelled) literals are labelled formulas.*
- *If F, F_1 and F_2 are labelled formulas and σ a label, then $\sigma.F$, $\neg F$, $F_1 \wedge F_2$, and $F_1 \vee F_2$ are labelled formulas.*
- *Nothing else is a labelled formula.*

We identify $\sigma_1.(\sigma_2.F)$ with $(\sigma_1\sigma_2).F$ and $\varepsilon.F$ with F. The length $l(\phi)$ and depth $d(\phi)$, and set of variables vars(ϕ) *are defined similarly as for modal formulas. We say that a set of formulas* contains *a label σ, in case it contains a formula $\sigma_0.F$ such that σ is a prefix of σ_0. A labelled formula is in* labelled NNF, *if all negation operators occur in front of a propositional variable.*

Intuitively, each '$*$' represents a '$\Box$', and each $c \in C$ represents a distinct '$\Diamond$'. The distinctness of possible worlds is made explicit by choosing different constants. Such a distinction is not required for the necessity operator. This is why we need only one wildcard symbol; it is understood that different occurrences of $*$ can be instantiated by different constants. We will manifest our intuition when we introduce the semantics of labelled formulas.

Throughout this paper, it will be convenient to refer to certain parts of a set of labelled formulas:

Definition 2. *Given a set S of labelled formulas, the* set generated by $c \in C \cup \{*\}$ *(written S_c) is the set $\{\sigma.F : c\sigma.F \in S$ or $*\sigma.F \in S\}$. The model S_σ generated by a label $\sigma = c_1 \ldots c_k$ is the set $((S_{c_1})_{c_2} \ldots)_{c_k}$.*

3.2 Free-Variable Models

As the next step towards defining the semantics of labelled formulas, we will now define their models.

Definition 3. *A set of labelled formulas of the form $\sigma.v$ where v is either $\top$ or a propositional variable, and σ is a (not necessarily ground) label, is called a* free-variable model.

We will now establish the relationship between Kripke models and free-variable models. The worlds of the Kripke models should be the ground labels explicitly or implicitly defined in the free-variable model, without including any unnecessary labels[5], while ensuring that they form a rooted tree with no missing intermediate nodes. The following definition, adapted from [3], affords this property:

Definition 4. *A ground label is called* realized (justified) by a model M, *if one of the following applies:*

- *ε is always realized by any model.*
- *$\sigma.c$ is realized by M, if σ is realized by M, and M contains a label of the form $\sigma_0.c$, where σ is an instance of σ_0.*

Proposition 1. *The graph G consisting of the ground labels realized by M, with directed edges (σ, σ') iff $\sigma' = \sigma.c$ with some $c \in C$, is a directed tree with root ε.*

Proof. By definition, ε is always realized, and we can easily see from Definition 4 that every prefix of a label realized by M is also realized by M. Therefore, there exists a directed path from ε to any other label in G. Finally, every label of the form $\sigma.c$ is the successor of exactly one label σ, which finishes the proof.

Given a free-variable model M, we construct a Kripke model as follows: Let $G = (W, R)$ be the graph as in Proposition 1, and $w_0 = \varepsilon$. Define $V(\sigma, v) =$ *true* if there exists a formula $\sigma_0.v$ in M such that σ is an instance of σ_0; set $V(\sigma, v) =$ *false* in all other cases. This completes the construction of the Kripke model (W, R, V) with root w_0, which we denote $K(M)$.

It is also possible to encode redundant information in free-variable formulas. For example, the formula $*.\phi$ states that ϕ must be true in all ground labels of length 1; if no such label is realized by the model in question, then $*.\phi$ is vacuously true, regardless of ϕ. Thus, even the formula $*.\bot$ is satisfied e.g. by the empty model. To identify labels which may never get instantiated, we introduce a counterpart to Definition 4:

Definition 5. *A label is* realizable *in a set of labelled formulas S (not necessarily a model), if one of the following applies:*

- *ε is always realizable.*
- *$\sigma.c$ is realizable in S if σ is.*
- *$\sigma.*$ is realizable in S if S contains a label of the form $\sigma_0.c$ for some constant c, and σ_0 and σ unify.*

Example 1. Let $C = \{1, 2\}$, and $V = \{p, q, r, s\}$, and consider the free-variable model $M = \{1.p, 1*.p, 2*.r, *1.q, *2.\top\}$. All labels in this model are realizable, and the realized ground labels are exactly the labels ε, 1, 2, 11, 12, 21, and 22. The Kripke model obtained from the above construction is displayed graphically in Figure 1, where x and $\overline{x}$ mean that x is assigned *true* and *false*, respectively.

[5] Including "too many" worlds sounds harmless but may in fact lead to unsoundness.

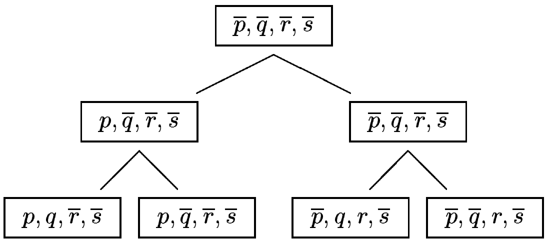
$\overline{p}, \overline{q}, \overline{r}, \overline{s}$
$p, \overline{q}, \overline{r}, \overline{s}$
$\overline{p}, \overline{q}, \overline{r}, \overline{s}$
$p, q, \overline{r}, \overline{s}$
$p, \overline{q}, \overline{r}, \overline{s}$
$\overline{p}, q, r, \overline{s}$
$\overline{p}, \overline{q}, r, \overline{s}$

Fig. 1. *A Kripke model for Example 1.*

3.3 Semantics of labelled formulas

With the definitions of the previous section, we can now formally define the semantics of labelled formulas.

Definition 6. *We say that the free-variable model M* satisfies, *or* is a model for *a labelled formula F, written $M \models F$, if the appropriate condition below applies (ϕ_1, ϕ_2 are labelled formulas, $x \in \mathcal{P}$, $c \in C$):*

$M \models \top$.	
$M \not\models \bot$.	
$M \models x$	*if* $\varepsilon.x \in M$.
$M \models \neg\phi$	*if* $M \not\models \phi$.
$M \models \phi_1 \wedge \phi_2$	*if* $M \models \phi_1$ *and* $M \models \phi_2$.
$M \models \phi_1 \vee \phi_2$	*if* $M \models \phi_1$ *or* $M \models \phi_2$.
$M \models c.\phi$	*if there exists a label* $c\sigma$ *in* M, *and* $M_c \models \phi$.
$M \models *.\phi$	*if for all* $c \in C$ *such that a label* $c\sigma$ *exists in* M, $M_c \models \phi$.

A formula is called *satisfiable* if it has a model, otherwise *unsatisfiable*. Two formulas ϕ_1 and ϕ_2 are *equivalent* if they are satisfied by the same models, and *contradictory* if $\phi_1 \wedge \phi_2$ is unsatisfiable. M satisfies a set of formulas if it satisfies every formula in it. All these terms are analogously defined for sets of formulas.

We state an important corollary, leaving its proof to the reader:

Corollary 1. *$M \models \sigma.\phi$ iff $M_{\sigma'} \models \phi$ for any realized ground instantiation σ' of σ, and the longest ground prefix of σ is realized in M.*

A free-variable model, viewed as a set of formulas, is always satisfied by itself, but also by any model M' with $M \subseteq M'$.

As a major difference to **K** where the possibility operator does not distribute over conjunctions, it's easy to prove that all labels do, including constants[6]:

Corollary 2. *For any label σ and any two labelled formulas ϕ_1, ϕ_2, we have the logical equivalence $\sigma.(\phi_1 \wedge \phi_2) \equiv \sigma.\phi_1 \wedge \sigma.\phi_2$.*

[6] However, labels do not distribute over *disjunctions*, an asymmetry which will be reflected in the definition of And/Or Normal form.

We are now ready to introduce satisfiability-preserving conversions from **K**-formulas to labelled formulas. We impose the (important) restriction that the **K**-formulas be in Negation Normal Form.

Theorem 1. *There exists an algorithm which, given a* **K***-formula F in NNF, takes linear time in the length of the input to construct a labelled formula ϕ in labelled NNF, such that any free-variable model M satisfies ϕ exactly when $K(M)$ is a Kripke model for F. Therefore, ϕ is satisfiable iff F is.*

Proof. Let us first state the algorithm. Given a formula F, we keep an array of labels $\lambda(S)$ for each subformula S of F, which we initialize to $\lambda(S) = \varepsilon$. The algorithm returns *label*(F). where *label*(S) is defined as follows:

- If S is $\top$, $\bot$, or a literal, return S.
- If S is $S_1 \circ S_2$ for any binary propositional connective, set $\lambda(S_1) = \lambda(S_2) = \lambda(S)$, and return *label*$(S_1) \circ$ *label*(S_2).
- If S is $\Box S_0$, set $\lambda(S_0) = \lambda(S)*$, and return $*.$*label*(S_0).
- If S is $\Diamond S_0$, find a constant $c \in C$ such that $\lambda(S)c$ does not unify with $\lambda(S')$ for any subformula S'. Then set $\lambda(S_0) = \lambda(S)c$, and return $c.$*label*(S_0).

It is easy to see that *label*(S) is in NNF provided S is, as no new negation symbols are introduced. Next, we prove the statement about the complexity of the algorithm. Observe that every subformula is accessed three times: to initialize $\lambda(S)$, to set $\lambda(S)$, and to determine *label*(S). In the fourth case above, we can avoid checking all subformulas of F for matching labels (which could result in quadratic run-time behaviour): we simply choose a fresh constant from C, each time the rule is encountered[7]. The correctness proof is left to the reader.

Another conversion, based on Corollary 2, will allow for a more concise tableau calculus. We define the *(labelled) And/Or Normal Form* (NF) as follows: All literals are in labelled And/Or NF; if further σ_i is a label, and ϕ_i is a disjunction of formulas in labelled And/Or NF for $i = 1, \ldots, n$, then $\phi = \bigwedge_{i=1}^{n} \sigma_i.\phi_i$ is in labelled And/Or NF. (This conjunction can be equivalently written as a set $\{\sigma_1.\phi_1, \ldots, \sigma_n.\phi_n\}$, a notation we will use frequently.)

Proposition 2. *Every labelled formula in Negation Normal Form can be converted to an equivalent formula in And/Or Normal Form. The transformation can be done in $O(d(\phi)l(\phi))$ time and space.*

Proof. The conversion simply consists of "factoring out" labels preceding conjunctions, which by Corollary 2 transforms formulas into equivalent formulas. To see the complexity result, observe that factoring out increases the size of each subformula in ϕ by at most the length of the longest possible label in the And/Or NF of ϕ, which is bounded by $d(\phi)$.

[7] Yet another feasible approach, in analogy to [3], is to assign a Gödelization of S_0 to c, so syntactically identical formulas are mapped to the same constant, which may lead to improvements if S_0 is complex.

Example 2. Converting the formula $\Diamond p \wedge \Diamond \neg p \wedge \Box(\Diamond q \wedge \Diamond \neg q) \wedge \Box((p \to \Box p) \wedge (\neg p \to \Box \neg p)) \wedge \Box \Box((p \vee r \vee s) \wedge (q \vee \neg s))$ successively into NNF, labelled NNF and labelled And/Or NF, yields the formula (written as a set) $S = \{1.p, 2.\neg p, *1.q, *2.\neg q, *.(\neg p \vee *.p), *.(p \vee *.\neg p), **.(p \vee r \vee s), **.(q \vee \neg s)\}$. (The constants 1 and 2 can be reused, since they do not occur in the same position as the first occurrences of 1 and 2.) The model shown in Example 1 satisfies this set.

3.4 Complementary Formulas and Clashes

We now state "obvious" cases for a set of formulas to be unsatisfiable, which the tableau algorithm will recognize as tableau closures:

Definition 7. *A* clash *is a formula of the form $\sigma.\bot$ with a realizable label σ, or a set of two formulas $\{\sigma_1.x, \sigma_2.\neg x\}$, where σ_1 and σ_2 are unifiable and (σ_1, σ_2) is realizable.*

This is a purely syntactic notion. Also, observe that only atomic formulas constitute a clash. In practice, recognizing contradictions between non-atomic formulas as early as possible is desirable [6]. Yet, the definition accurately reflects the semantic notion of unsatisfiability and is strong enough for our purposes:

Proposition 3. *If a set of formulas contains a clash, it is unsatisfiable. Conversely, a* disjunction-free *formula in And/Or NF (written as a set) is satisfiable, provided it contains no clash. A model is obtained by removing all occurrences of the form $\sigma.\bot$ and replacing negative literals $\sigma.\neg x$ with $\sigma.\top$.*

Proof. The first half of the proof is obtained by induction over the length of the labels of the formula(s) involved in the clash. For the converse, it is easy to show that the model constructed as above for a clash-free formula S satisfies each formula in S. Note that contradictory literals in S can only occur with labels which are not realizable, which does not affect the correctness of the model.

Notice that we cannot simply remove negative literals from S, as their labels, if realizable, are an essential part of the model; only labels which are already realizable in other formulas in S may be dropped.

4 The Free-Variable Tableau Algorithm

The only step remaining in the model construction is to eliminate the disjunctions in the And/Or Normal Form. Our tableau calculus accomplishes just that. We specify it here in its most basic form. Our definition of semantic tableaux follows the standard set forth e.g. in [7]. Let S_0 be a set of formulas in labelled And/Or NF. A *semantic tableau* for S_0 is a tree whose nodes are sets of formulas, with root node S_0, constructed recursively as follows: The singleton tree $(\{S_0\}, \emptyset)$ is a tableau. If $\mathcal{T}$ is a tableau, then we call the paths from root S_0 to leaves in $\mathcal{T}$ *branches.* (We identify a branch with the set union of all its nodes.)

A tableau $\mathcal{T}'$ is created from $\mathcal{T}$ by *applying* the following *rule* to a branch B in $\mathcal{T}$ (also called *tableau expansion*):

$$\frac{\sigma.(\beta_1 \vee \cdots \vee \beta_r)}{\begin{array}{c|c|c} \sigma'\sigma_{1,1}.\phi_{1,1} & \cdots & \sigma'\sigma_{r,1}.\phi_{r,1} \\ \cdots & & \cdots \\ \sigma'\sigma_{1,k_1}.\phi_{1,k_1} & & \sigma'\sigma_{r,k_r}.\phi_{r,k_r} \end{array}}\,, \tag{1}$$

That is: If S is the leaf node of B, and the *premise* $\sigma.(\beta_1 \vee \cdots \vee \beta_r)$ is a formula in B, where $\beta_j = \{\sigma_{1,1}.\phi_{1,1}, \ldots, \sigma_{1,k_1}.\phi_{1,k_1}\}$ $(j = 1, \ldots, r)$, then the tree obtained from $\mathcal{T}$ by appending the nodes (*conclusions*) $\{\sigma'\sigma_{1,1}.\phi_{1,1}, \ldots, \sigma'\sigma_{1,k_1}.\phi_{1,k_1}\}, \ldots,$ $\{\sigma'\sigma_{r,1}.\phi_{r,1}, \ldots, \sigma'\sigma_{r,k_r}.\phi_{r,k_r}\}$ to S, with σ' being a realized ground instance of σ, is a tableau for S_0. This is where the effort of converting to And/Or NF pays off: In an ordinary modal **K** tableau system, even when normal forms are used, at least four different rules are needed, one for each propositional and modal operator (except $\neg$). In our calculus, only one rule to break up disjunctions remains; the others are implicitly contained in the conversion.

A branch S is *closed*, if it contains a clash, otherwise it is *open*. Finally, a tableau is *closed*, if all its branches are closed, and *open* otherwise.

Note that the same tableau rule could be applied to the same premise on a branch arbitrarily often, thus producing the same conclusions over and over again. A tableau expansion like this would never terminate, which we want to avoid. Therefore, we allow a rule application to the current branch B only if it is *admissible* to B, that is, when the premise is on the branch, but none of the conclusions is a set of instances of formulas already on B. We call a branch B *complete*, provided that no rule has any instances admissible to B.

Clearly, the tableau rule applies exactly to all formulas on the current branch which are not disjunction-free. Furthermore, every label occurring in S_0, or anywhere on any branch, has only finitely many realized ground instances, so each disjunction can be expanded only finitely many times according to the tableau rule. Observe further that each rule application produces only formulas in And/Or NF; each consequence contains fewer disjunctions than the premise, and none of the labels can become longer than $d(S_0)$. Since we start out with finitely many disjunctions, even accounting for nested ones, each branch of the tableau will become complete after finitely many rule applications. Therefore, we can state our first result:

Proposition 4. *Expansion of a free-variable tableau for any set S_0 in And/Or NF terminates after finitely many admissible rule applications in a tableau in which every branch is either closed or complete.*

4.1 Soundness and Completeness

Let us now show that the tableau calculus is correct, given our semantics for labelled formulas:

Proposition 5. *The application of rule (1) is semantically correct, that is, every model for an instance of its premise is a model for one of its conclusions.*

Proof. Let M be a model for the formula $\sigma.(\beta_1 \vee \ldots \vee \beta_r)$. By Corollary 1, $M_{\sigma'} \models (\beta_1 \vee \ldots \vee \beta_r)$ for any realized ground instance σ' of σ. Therefore, by Definition 6, $M_{\sigma'} \models \beta_i$, and further $M_{\sigma'} \models \sigma_{i,j}.\phi_{i,j}$ for all $j = 1, \ldots, k_i$, for some i. Once again by Corollary 1, we conclude that M satisfies all the formulas in one of the conclusions.

Proposition 6. *Let T be a tableau for S_0, and T' a tableau generated from T by a single rule application. Let B and B' be two corresponding branches in T and T', respectively. Then the set of formulas on B is satisfiable if and only if the set of formulas on B' is satisfiable. The two branches are satisfied by exactly the same models.*

Proof. If the branch is untouched by the tableau expansion, there is nothing to show. Otherwise, one of the formulas on B is used as the premise for rule (1). Therefore, a model for B also satisfies this premise, and by Proposition 5, it also satisfies one of the conclusions. The converse is trivial, as the set of formulas on B is a subset of B', whence any model of B' is a model of B.

Remembering Proposition 3 which said that a set of formulas containing a clash is unsatisfiable, Proposition 6 allows us to go back up along the expansion steps of a closed tableau to the root tableau S_0 and conclude:

Corollary 3. *(Soundness) If a tableau for a set S_0 is closed, then S_0 is unsatisfiable.*

To show completeness and decidability, let us assume that a tableau for S_0 has been expanded completely, meaning that every branch is either closed or complete. This is possible in finitely many steps, according to Proposition 4.

Proposition 7. *The set of formulas on a complete branch is satisfiable if and only if the set of all disjunction-free formulas on the branch is satisfiable. A model of this set is also a model of the entire branch.*

Proof. Note that one direction of this proof is trivial as before. We will only sketch the proof of the other direction which makes use of a variant of so-called *Hintikka sets* [8]. These sets are defined so that for any formula of the shape $\sigma.(\beta_1 \vee \ldots \vee \beta_r)$ in the set, for each realized ground instance σ' of σ, at least one disjunct $\sigma'.\beta_i$ is also contained in the set. Now if M is a model which satisfies all the "elementary", disjunction-free formulas in the set, then by Proposition 5 M also satisfies the "compound" formulas. Finally, it is easy to show that the formulas on a complete branch are a Hintikka set as defined, from which the proof follows.

Now recall Proposition 3, which says that any unsatisfiable set of disjunction-free formulas contains a clash. Putting this result and Propositions 6 and 7 together, we obtain that every branch of a tableau of an unsatisfiable set S_0 contains a clash, which shows:

Corollary 4. *(Completeness) If S_0 is unsatisfiable, then S_0 has a closed tableau.*

Soundness and completeness results prove the correctness of a tableau calculus. In practice however, we wish to decide whether a given S_0 is satisfiable, without having to try all possible tableaux for S_0. To this end, we expand the tableau by admissible expansion rules only, until either all branches are closed (in which case we know S_0 is unsatisfiable) or we find a complete open branch, which means that the subset of all disjunction-free formulas is clash-free and hence satisfiable; Proposition 3 lets us obtain a model which, by Propositions 7 and 6, is also a model for S_0. This shows:

Corollary 5. *For any set S_0 of free-variable formulas in And/Or NF, starting with a tableau S_0 and applying rule (1) in any order, until all branches are closed or one branch is complete, provides a finite decision procedure for S_0. Each complete open branch of any tableau for S_0 provides a model for S_0.*

Finally, we wish to show the correctness of the calculus with regard to the original logic **K**. Consider an arbitrary **K**-formula F. First, F can be converted into an equivalent formula F' in NNF. Then Theorem 1 and its proof provided an algorithm to convert F' into a labelled formula ϕ in NNF, which is satisfiable if and only F' is. By Corollary 2, ϕ can be converted to an equivalent formula ϕ' in And/Or NF, to which the tableau algorithm can be applied, which decides (in finite time) whether ϕ', and hence F, is satisfiable. If M is a model offered by a complete open branch for ϕ', we can step back through the array of theorems and conclude that $K(M)$ is a Kripke model for F. Hence:

Theorem 2. *The labelled formula formalism with the conversions as specified in Section 3.3, and free-variable tableau algorithm as specified in Section 4, provide a sound and complete decision and model finding procedure for* **K**.

5 Comparison to Previous Approaches

The use of variables in labelled tableaux for modal logics is nothing new. Most notably, the already mentioned work by Beckert and Goré [2, 3] is a comprehensive treatment of free-variable tableaux for various modal logics. Although our own approach has been derived independently, it can be viewed as a variation of Beckert and Goré's free-variable tableaux. In the following, we will highlight the connections and important differences between the two approaches. For the purpose of this survey, familiarity with the terminology in [3] is assumed.

- In [3], two notions of variables are used: *universal* and *free variables.* Universal variables arise from the necessity operator; they can stand in for any successor world (constant). Free variables arise from branching on a disjunction prefixed by universal variables. They are no longer universal in that occurrences of the same free variable in different branches must be instantiated with the same constant to preserve soundness.
  The wildcard symbol in our approach represents universal variables. Since they can be arbitrarily renamed, we can represent them by an anonymous

variable $*$, with the understanding that two occurrences of $*$ can be instantiated differently.

Our approach does not provide any equivalent to free variables. In rule (1), the label σ must be instantiated by a realized *ground* label. We thus lose the power of reasoning with several instances of the disjunction at once; in fact, one can show that our free-variable tableau requires the same number of branches as an equivalent tableau with ground labels only. (However, the branches themselves can still be represented more compactly, thanks to occurrences of $*$ in literals.) In those cases where other formulas on the branch do not require a ground instantiation and renaming, the approach in [3] is strictly stronger.

– We have given a rigid definition of *free-variable models* which is novel. The implementation of **LeanK** does not return a satisfying model, although it is claimed that such a model is easy to obtain [2]. A **K**-satisfiable tableau (Definition 3.10 in [3]) can rightfully be viewed as a model. However, obtaining a Kripke model from a **K**-satisfiable tableau is not straightforward, potentially involving different branches of the tableau for different instantiations of the free variables. The advantage of abandoning the use of free variables is that all labelled literals pertaining to a free-variable model can be found on one single branch, just as in our approach.

  Free-variable models are a powerful representation format of Kripke models. For example, in a model where all labels of length n consisting of the constants 0 and 1 are represented, the formula $*^n.p$ represents truth assignments of p in 2^n worlds. Furthermore, free-variable models are consistent by definition, and they represent a unique Kripke model. We acknowledge, though, that our basic tableau algorithm does not adequately exploit this expressive power, since all occurrences of $*$ in labels of disjunctions are replaced by constants during the proof search.

– In our approach, a formula in NNF is pre-processed in two steps: in-situ replacement of all modal operators by labels (Theorem 1), and conversion to And/Or Normal Form (Proposition 2). The proper tableau calculus then requires only one expansion rule, which is really a combination of the α-, β-, and ν-rules of the classical modal tableau calculus [7] and can be simulated by a classical tableau with fixed prioritization of rule application, as β-rules are delayed until no other rule can be applied. This prioritization is preferrable as it minimizes branching. We have effectively removed the nondeterminism in choosing a suitable tableau rule; we only need to choose the next instance of a disjunction to branch on, as well as the order of subbranches to expand. A further benefit of representation in And/Or Normal form is that it will simplify the implementation of the calculus. Subformulas are of exactly the same form and can be represented by the same data structures as the main formula. Any techniques used on the main formula, such as tableau expansion, simplification, caching etc., can be used on subformulas as well, thus enhancing the effect of these respective techniques.

  The tableau calculus in [3] can be modified to simulate our preprocessing step by allowing rule applications in subformulas and deferring application of the disjunctive rule until all other rule applications have been performed.
- The calculus in [3] requires a substitution rule and a closure rule. The former is needed to ensure that labels involved in clashes are realizable (or justified); it is not necessary in the absence of free variables. The closure rule requires existence of a justified *ground* instantiation of two labels in order to close a branch, whereas we require that their mgu be realizable. It can be shown that these two notions are equivalent[8].
- In our approach, we prefer using ε as label for the initial formula; in [3], this label is 1. For all modal logics in which the rooted tree model property holds, this is a legitimate choice, and the difference is solely a matter of taste. The empty label fits our calculus better, since labels inside formulas can also be empty.
- The diamond rule in [3] uses a Gödelization of the formula A as label to represent $\Diamond A$. Hence, occurrences of syntactically identical formulas would be represented by the same label, thus avoiding duplicated efforts in expanding this formula. Our preprocessing algoritm (Theorem 1) could do the same; however, other low-cost techniques such as lexical normalization [15] detect a much larger subclass of equivalent formulas.

To summarize, the only fundamental difference of the free-variable tableau calculus in [3] compared to ours is its use of free variables. This makes it strictly more powerful in theory. However, it is not clear that free variables provide an advantage in practice, especially since the need for a substitution rule creates additional nondeterminism, and since satisfying models are less straightforward to obtain from satisfiable tableaux. The drawback of not having free variables to represent several instances may be offset in practice by techniques like model caching, model merging or absorption.

6 Implementability and Conclusions

Despite having only one tableau rule and being easy to prove sound and complete, the basic calculus does not give rise to a powerful algorithm at all. This is because the label of a disjunctive formula must be instantiated by a ground label, which is tantamount to creating all the branches of an equivalent classical tableau. Furthermore, we need to test for admissibility of the tableau rule to each instance of a premise, in order to obtain a finite decision procedure. To solve the second problem, we can simply maintain a list $\mathrm{app}(\sigma.\phi)$ of (not necessarily ground) instances of σ to which a disjunction $\sigma.\phi$ has already been applied. Application of rule (1) is restricted to realized ground instances which are not instances of any element in $\mathrm{app}(\sigma.\phi)$. If no disjunction has any such instances, the current branch is complete.

[8] Indeed, **LeanK** first unifies the labels of two clashing literals and then checks if the result is realizable.

To address the first problem, we must reduce the number of rule applications, since each of them branches. We expect simplification to be a helpful approach. For example, if a formula $*1.p$ exists on the current branch, we need not branch on the disjunction $**.(p \vee q)$ for instances of $*1$, since a stronger formula already holds in these instances. So we simply add $*1$ to the labels in $\mathrm{app}(**.(p \vee q))$. We claim that a simplification algorithm can be devised which will greatly reduce branching in Ladner encodings [18] of QBF problems. A careful treatment of simplification including an implementation to prove this claim is currently in progress and will be presented in a future publication.

Beyond simplification, optimization strategies from [3, 14, 15] can be adapted, such as backjumping, model caching, heuristics, model merging, factorization etc. Caching in particular promises to work favourably with labelled formulas, which we also leave as future work to demonstrate.

Finally, we must decide on the order of application of the tableau rule. A depth-first approach seems most feasible; however, the And/Or Normal Form allows us to break this default order and branch on subformulas first, if for example, this subformula likely leads to closure. A good heuristic to find these cases is essential.

To summarize, the free-variable formalism presented here offers these main advantages:

- The different steps of satisfiability finding in **K** are cleanly separated, and their complexities can be highlighted: First, **K**-formulas are converted into free-variable formulas in And/Or NF in linear time and space (provided the modal depth is bounded); thereby we create a "template" of a Kripke tree. Then the tableau algorithm, consisting of only one rule, branches on the disjunctions, taking exponential space and double-exponential time in the basic version, determining variable assignments in the form of labelled literals. To close a tableau branch, we check a set of labelled literals for clashes, which is closely related to satisfiability in the description logic $\mathcal{ALE}$ and NP-complete [5].
- Implementations of the classical labelled tableau calculus [7] typically search breadth-first or depth-first through the worlds (ground labels) of a model. In a free-variable tableau algorithm, we can find clashes deep in the model tree und thus close a branch before time is wasted expanding consistent formulas with shorter labels.
- We have given a definition of free-variable models which allows a more compact representation of satisfying models than is possible with Kripke models.
- To exploit the expressive power of free-variable models and improve the performance of the tableau algorithm, simplification and other techniques can be devised, reducing the need to instantiate labels with $*$ placeholders.

References

1. M. d'Agostino, D. M. Gabbay, A. Russo. Grafting Modalities onto Substructural Implication Logics, *Studia Logica*, 59, pp. 65–102, 1997.

2. B. Beckert, R. Goré. System Description: LeanK 2.0: Free Variable Tableaux for Propositional Modal Logics: Decision Procedures, in: CADE-15: Proceedings of the International Conference on Automated Deduction, pp. 51–55, Springer, LNAI 1421, 1998.
3. B. Beckert, R. Goré. Free Variable Tableaux for Propositional Modal Logics. : *Studia Logica*, 69:59–96, 2001.
4. B. Chellas. Modal Logic—an Introduction. Cambridge University Press, 1980.
5. F. M. Donini, B. Hollunder, M. Lenzerini, A. M. Spaccamela, D. Nardi, W. Nutt, The Complexity of Existential Quantification in Concept Languages. *Artificial Intelligence*, 53:309–327, 1992.
6. F.M. Donini, F. Massacci. EXPTIME tableaux for ALC. *Artificial Intelligence*, 124(1):87–138, 2000.
7. M. C. Fitting, Proof Methods for Modal and Intuitionistic Logics. D. Reidel Publishing, Dordrecht, Holland, 1983.
8. M. C. Fitting. First-order Logic and Automated Theorem Proving. Springer, New York, Berlin, Heidelberg, 1996.
9. R. Goré. Tableau methods for Modal and Temporal Logics. In: M. d'Agostino, D. M. Gabbay, R. Hähnle, J. Posegga (eds.), *Handbook of Tableau Methods*, Kluwer Academic Publishers, Dordrecht/Boston/London, pp. 297–396, 1999.
10. G. Governatori. Labelled Tableaux for Multi-Modal Logics. In: *Theorem Proving with Analytic Tableaux and Related Methods (TABLEAUX 1995)*, pp. 79–94, Springer, LNAI 918, 1995.
11. V. Haarslev, R. Möller. Consistency Testing: the RACE Experience. In: [21], pp. 57–61, 2000.
12. J. Happe. The ModProf Theorem Prover. In: *Proc. of the Int'l Joint Conference on Automated Reasoning (IJCAR 2001)*. Springer, LNAI, pp. 349–353, 2001.
13. J. Happe, A Subsumption-Aided Caching Technique. In: *Issues in the Design and Experimental Evaluation of Systems for Modal and Temporal Logics (IJCAR 2001 Workshop)*, Tech. Report DII 14/01, Università degli Studi di Siena, Italy, pp. 49-57, 2001.
14. I. Horrocks. Optimising Tableaux Decision Procedures for Description Logics. Ph.D. thesis, Univ. of Manchester, U.K., 1997.
15. I. Horrocks, P. F. Patel-Schneider. Optimising Description Logic Subsumption. *Jnl. of Logic and Computation*, 9(3):267–293, 1999.
16. I. Horrocks, P. F. Patel-Schneider. The Generation of DAML+OIL. In: *Working Notes of the 2001 Int'l Description Logics Workshop (DL-2001)*, pp. 30-35, Stanford, CA., 2001.
17. S. A. Kripke. Semantical Analysis of Modal Logic I: Normal Propositional Calculi. In: *Zeitschrift für mathematische Logik und Grundlagen der Mathematik* 9:67–96, 1963.
18. R. E. Ladner. The Computational Complexity of Provability in Systems of Modal Propositional Logic. SIAM Jnl. of Computing 6(3), pp. 467–480, 1977.
19. F. Massacci. Simplification: A General Constraint Propagation Technique for Propositional and Modal Tableaux. *Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 1998)*, pp. 217–231, Springer, LNAI 1397, 1998.
20. F. Massacci. Design and Results of TANCS-2000 Non-classical (Modal) Systems Comparison. In: [21], pp. 52–56, 2000.
21. R. Dyckhoff (ed.). *Automated Reasoning with Analytic Tableaux and Related Methods — Proceedings of the Int'l Conference (TABLEAUX 2000)*. Springer, LNAI 1847, 2000.