Free Variable Tableaux for Propositional Modal Logics

Bernhard Beckert*

Imperial College
Department of Computing
180 Queen's Gate, London SW7, England
beckert@ira.uka.de
http://i12www.ira.uka.de/~beckert

Rajeev Goré

Automated Reasoning Project
Australian National University
Canberra, ACT, 0200, Australia
rpg@arp.anu.edu.au
http://arp.anu.edu.au/

Abstract. We present a sound, complete, modular and *lean* labelled tableau calculus for many propositional modal logics where the labels contain "free" and "universal" *variables*. Our "lean" Prolog implementation is not only surprisingly short, but compares favourably with other considerably more complex implementations for modal deduction.

1 Introduction

Free variable semantic tableaux are a well-established technique for first-order theorem proving—both theoretically and practically. Free variable quantifier rules [19, 7] are crucial for efficiency since free variables act as a meta-linguistic device for tracking the eigenvariables used during proof search.

Traditional tableau-based theorem provers developed during the last decade for first-order logic have been complex and highly sophisticated, typified by systems like Setheo [16] and ${}_3T^AP$ [2]. On the other hand, free variable tableaux, and their extensions like universal variable tableaux, have been used successfully for *lean* Prolog implementations, as typified by leanTAP [3]. A "lean" implementation is an extremely compact (and efficient) program that exploits Prolog's built-in clause indexing scheme and backtracking mechanisms instead of relying on elaborate heuristics. Such compact lean provers are much easier to understand than their more complex stablemates, and hence easier to adapt to special needs.

Simultaneously, Kanger's meta-linguistic indices for non-classical logics [15] have been generalised by Gabbay into Labelled Deductive Systems [10]. And Massacci [17] and Russo [20] have recently shown the utility of using *ground* labels for obtaining *modular* modal tableaux and natural deduction systems (respectively); see [11] for an introduction to labelled modal tableaux.

By allowing labels to contain free (and universal) variables, we obtain *efficient* and *modular* tableaux systems for all the 15 basic *propositional* modal logics. Furthermore, our leanTAP style implementation compares favourably with existing fast implementations of modal tableau systems like LWB [13].

* On leave from University of Karlsruhe, Institute for Logic, Complexity and Deduction Systems, D-76128 Karlsruhe, Germany.

Our object language uses *labelled formulae* like $\sigma : A$, where σ is a label and A is a formula, with intuitive reading "the possible world σ satisfies the formula A"; see [6, 18, 11] for details. Thus, $1 : \Box p$ says that the possible world 1 satisfies the formula $\Box p$. Our box-rule then reduces the formula $1 : \Box p$ to the labelled formula $1.(x) : p$ which contains the *universal* variable x in its label and has an intuitive reading "the possible world $1.(x)$ satisfies the formula p". Since different instantiations of x give different labels, the labelled formula $1.(x) : p$ effectively says that "all successors of the possible world 1 satisfy p", thereby capturing the usual Kripke semantics for $\Box p$ (almost) exactly. But the possible world 1 may have *no* successors; so we enclose the variable in parentheses and read $\sigma : A$ as "for all instantiations of the variables in σ, if the world corresponding to that instantiation of σ exists then the world satisfies the formula A".

Similar approaches using labels containing variables have been explored by Governatori [12] and D'Agostino et al. [5]. But D'Agostino et al. relate the labels to modal algebras, instead of to first-order logic as we do. And whereas Governatori uses string unification over labels to detect complementary formulae, we use Prolog's matching, since string unification cannot be implemented in a lean way. Our variables are of a simpler kind: they capture all *immediate* children of a possible world (in a rooted tree model), but do not capture *all R-successors*; see [17, 11]. As a consequence, we can make extensive use of Prolog features like unification and backtracking in our implementation. Note, however, that a non-lean extension of our calculi using string unification is perfectly feasible.

The following techniques, in particular, are crucial:

Free variables: Applying the traditional ground box-rule requires guessing the correct eigenvariables. Using (free) variables in labels as "wildcards" that get instantiated "on demand" during branch closure allows more intelligent choices of these eigenvariables. To preserve soundness for worlds with no R-successors, variable positions in labels must be conditional.

Universal variables: Under certain conditions, a variable x introduced by a formula like $\Box A$ is "universal" in that an instantiation of x on one branch need not affect the value of x on other branches, thereby localising the effects of a variable instantiation to one branch. The technique entails creating and instantiating local duplicates of labelled formulae instead of the originals.

Finite diamond-rule: Applying the diamond-rule to $\Diamond A$ usually creates a new label. By using (a Gödelisation of) the formula A itself as the label instead, we guarantee that only a finite number of different labels (of a certain length) are used in the proof. In particular, different (identically labelled) occurrences of $\Diamond A$ generate the same unique label.

The paper is structured as follows: In Sections 2 and 3 we introduce the syntax and semantics of labelled modal tableaux. In Section 4 we introduce our calculus, formalise its soundness and completeness (full proofs can be found in [1]), and present an example. In Section 5 we describe our implementation and present experimental results; and in Section 6 we present our conclusions and discuss future work.

2 Syntax

The formulae of modal logics are built-in the usual way; see [11]. To reduce the number of tableau rules and the number of case distinctions in proofs, we restrict all considerations to implication-free formulae in negated normal form (NNF); thus negation signs appear in front of primitive propositions only. Using NNF formulae is no real restriction since *every* formula can be transformed into an equivalent NNF formula in linear time.

Labels are built from natural numbers and variables, with variables intended to capture the similarities between the $\forall$ quantifier of first-order logic and the $\Box$ modality of propositional modal logic. However, whereas first-order logic forbids an empty domain, the $\Box$ modality tolerates possible worlds with no successors.[2] To capture this (new) behaviour, variable positions in labels are made "conditional" on the existence of an appropriate successor by enclosing these conditional positions in parentheses.

Definition 1. Let Vars be a set of variables, and let $\mathbf{N}$ be the set of natural numbers. Let x, y, z range over arbitrary members of Vars, let n and m range over arbitrary members of $\mathbf{N}$, and let l range over arbitrary members of $\text{Vars} \cup \mathbf{N}$. Then, the string 1 is a **label**; and if σ is a label, then so are $\sigma.m$ and $\sigma.(l)$. The **length** of a label σ is the number of dots it contains plus one, and is denoted by $|\sigma|$. The constituents of a label σ are called **positions** in σ and terms like "the 1st position" or "the n-th position" are defined in the obvious way. A position is **conditional** if it is of the form (l), and a label is conditional if it contains a conditional position. By $\text{ipr}(\sigma)$ we mean the set of all non-empty **initial prefixes** of a label σ, excluding σ itself. A label is **ground** if it consists of (possibly conditional) members of $\mathbf{N}$ only. Let $\mathcal{L}$ be the set of all ground labels.

When dealing with ground labels, we often do not differentiate between the labels $\sigma.n$ and $\sigma.(n)$, and we use $\sigma.[n]$ to denote that the label may be of either form. Note also that $\sigma.x$ (parentheses around x omitted) is not a label: the parentheses mark the positions that contain variables, or that used to contain variables before a substitution was applied.

Definition 2. A set Γ of labels is **strongly generated** if: (a) there is some (root) label $\rho \in \Gamma$ such that $\rho \in \text{ipr}(\sigma)$ for all $\sigma \in \Gamma \setminus \{\rho\}$; and (b) $\sigma \in \Gamma$ implies $\tau \in \Gamma$ for all $\tau \in \text{ipr}(\sigma)$.

Since we deal with mono-modal logics with semantics in terms of rooted frames (see Section 3), we always assume that our labels form a strongly generated set with root $\rho = 1$. In any case, our definition of labels guarantees that all our labels begin with 1, and it is easy to see that the labels that appear in any of our tableaux are strongly generated.

[2] To that extent, modal logics are similar to free logic, i.e., first-order logic where the domains of models may be empty [4].

Definition 3. A **labelled tableau formula** (or just tableau formula) is a structure of the form $X : \Delta : \sigma : A$, where X is a subset of $\mathrm{Vars} \cup \mathbf{N}$, Δ is a set of labels, σ is a label, and A is a formula in NNF. If the set Δ is empty, we use $X : \sigma : A$ as an abbreviation for $X : \emptyset : \sigma : A$. A tableau formula $X : \Delta : \sigma : A$ is **ground**, if σ and all labels in Δ are ground. If $\mathcal{F}$ is a set of labelled tableau formulae, then $\mathrm{lab}(\mathcal{F})$ is the set $\{\sigma \mid X : \Delta : \sigma : A \in \mathcal{F}\}$.

The intuitions behind the different parts of our "tableau formulae" are as follows: The fourth part A is just a traditional modal formula. The third part σ is a label, possibly containing variables introduced by the reduction of $\Box$ modalities. If the label σ is ground, then it corresponds to a particular path in the intended rooted tree model; for example, the ground label 1.1.1 typically represents the leftmost child of the leftmost child of the root 1. If σ contains variables, then it represents all the different paths (successors) that can be obtained by different instantiations of the variables, thereby capturing the semantics of the $\Box$ modalities that introduced them. Our rule for splitting disjunctions allows us to retain these variables in the labels of the two disjuncts, but because $\Box$ does not distribute over $\vee$, such variables then lose their "universal" force, meaning that these "free" variables can be instantiated only *once* in a tableau proof. We use the first component X to record the variables in the tableau formula ϕ that are "universal", meaning that ϕ can be used multiply in the same proof with different instantiations for these variables. The free variables in ϕ (that do not appear in X) can be used with only one instantiation since they have been pushed through the scope of an $\vee$ connective. The second part Δ, which can be empty, has a significance only if our calculus is applied to one of the four logics **KB**, **K5**, **KB4**, and **K45** (that are non-serial, but are symmetric or euclidean, see Section 3). It is empty for the other logics. The intuition of Δ is that the formula A has to be true in the possible world called σ only if the labels in Δ name legitimate worlds in the model under consideration. This feature has to be used, if (a) rule applications may shorten labels, which is the case if the logic is symmetric or euclidean, and (b) the logic is non-serial and, thus, the existence of worlds is not guaranteed. The set Δ can contain both universal and free variables, and some of them may appear in σ.

Definition 4. Given a tableau formula $\phi = X : \Delta : \sigma : A$, $\mathrm{Univ}(\phi) = X$ is the set of **universal variables** of ϕ, while $\mathrm{Free}(\phi) = \{x \text{ appears in } \sigma \text{ or } \Delta \mid x \notin X\}$ is the set of **free variables** of ϕ. These notions are extended in the obvious way to obtain the sets $\mathrm{Free}(\mathcal{T})$ and $\mathrm{Univ}(\mathcal{T})$ of free and universal variables of a given tableau $\mathcal{T}$ (see Def. 5).

Definition 5. A **tableau** is a (finite) binary tree whose nodes are tableau formulae. A **branch** in a tableau $\mathcal{T}$ is a maximal path in $\mathcal{T}$.[3] A branch may be marked as being **closed**. If it is not marked as being closed, it is **open**. A tableau branch is **ground** if every formula on it is ground, and a tableau is ground if all its branches are ground.

[3] Where no confusion can arise, we identify a tableau branch with the set of tableau formulae it contains.

Since we deal with propositional modal logics, notions from first-order logic like variables and substitutions are needed only for handling meta-linguistic notions like the accessibility relation between worlds. Specifically, whereas substitutions in first-order logic assign terms to variables, here they assign numbers or other variables (denoting possibles worlds) to variables.

Definition 6. A **substitution** is a (partial) function $\mu : \mathrm{Vars} \to \mathbf{N} \cup \mathrm{Vars}$. Substitutions are extended to labels and formulae in the obvious way. A substitution is **grounding** if its range is the (whole) set Vars and its range is $\mathbf{N}$; that is, if it maps all variables in Vars to natural numbers. A substitution is a **variable renaming** if its range is Vars, and it replaces distinct variables by other distinct variables only. The **restriction** of a substitution μ to a set X of variables is denoted by $\mu_{|X}$.

Definition 7. Given a tableau $\mathcal{T}$ containing a tableau formula $X : \Delta : \sigma : A$, a tableau formula $X' : \Delta' : \sigma' : A$ is a **$\mathcal{T}$-renaming** of $X : \Delta : \sigma : A$ if there is a variable renaming μ such that $X' : \Delta' : \sigma' : A = (X : \Delta : \sigma : A)\mu$, and every variable introduced by μ is new to the tableau $\mathcal{T}$.

3 Semantics

To save space, we assume familiarity with Kripke semantics for propositional normal modal logics; see [11] for details of any undefined terms. A world $w \in W$ is **idealisable** if it has a successor in W. To illustrate the modularity of our method we concentrate on the five basic axioms known as (T) $\Box A \to A$, (D) $\Box A \to \Diamond A$, (4) $\Box A \to \Box\Box A$, (5) $\Diamond A \to \Box\Diamond A$, (B) $A \to \Box\Diamond A$, and the 15 extensions of the basic propositional normal modal logic **K** obtained as shown in the first two columns of Table 1. The following properties of the reachability relation R characterise these axioms, (T): reflexivity, (D): seriality, (4): transitivity, (5): euclideanness, and (B): symmetry; see [11] for details. We therefore obtain:

Definition 8. For any logic **L** from Table 1, $\langle W, R\rangle$ is an **L**-frame if each axiom of **L** is valid in $\langle W, R\rangle$. A model $\langle W, R, V\rangle$ is an **L**-model if $\langle W, R\rangle$ is an **L**-frame.

It is also well-known that finer characterisation results for these logics can be given in terms of *finite rooted tree* frames; see [11] for details. These results are built into the following definition of **L**-accessibility which views a set of strongly generated ground labels as a tree with root ρ where $\sigma.[n]$ is an immediate child of σ (hence the name "strongly generated").

Definition 9. Given a logic **L** and a set Γ of strongly generated ground labels with root $\rho = 1$, a label $\tau \in \Gamma$ is **L-accessible** from a label $\sigma \in \Gamma$, written as $\sigma \rhd \tau$, if the conditions set out in Table 1 are satisfied. A label $\sigma \in \Gamma$ is an **L-deadend** if no $\tau \in \Gamma$ is **L**-accessible from σ.

The following lemma (see [11] for a proof) shows that the **L**-accessibility relation $\rhd$ on labels has the properties like reflexivity, transitivity, etc. that are appropriate for **L**-frames.

Lemma 10. *If Γ is a strongly generated set of ground labels with root $\rho = 1$, then $\langle \Gamma, \rhd \rangle$ is an* **L***-frame.*

Logic	Axioms	τ is **L**-accessible from σ	Logic	Axioms	τ is **L**-accessible from σ
K	(K)	$\tau = \sigma.[n]$	**KT**	(KT)	$\tau = \sigma.[n]$ or $\tau = \sigma$
KB	(KB)	$\tau = \sigma.[n]$ or $\sigma = \tau.[m]$	**K4**	(K4)	$\tau = \sigma.\theta$
K5	(K5)	$\tau = \sigma.[n]$, or $\lvert\sigma\rvert \geq 2$ and $\lvert\tau\rvert \geq 2$	**K45**	(K45)	$\tau = \sigma.\theta$, or $\lvert\sigma\rvert \geq 2$ and $\lvert\tau\rvert \geq 2$
KD	(KD)	**K**-condition, or σ is a **K**-deadend and $\sigma = \tau$	**KDB**	(KDB)	**KB**-condition, or $\lvert\Gamma\rvert = 1$ and $\sigma = \tau = 1$
KD4	(KD4)	**K4**-condition, or σ is a **K**-deadend and $\sigma = \tau$	**KD5**	(DK5)	**K5**-condition, or $\lvert\Gamma\rvert = 1$ and $\sigma = \tau = 1$
KD45	(KD45)	**K45**-condition, or $\lvert\Gamma\rvert = 1$ and $\sigma = \tau = 1$	**KB4**	(KB4)	$\lvert\Gamma\rvert \geq 2$
B	(KTB)	$\tau = \sigma$, or $\tau = \sigma.[n]$, or $\sigma = \tau.[m]$	**S4**	(KT4)	$\tau = \sigma.\theta$ or $\tau = \sigma$
S5	(KT5)	all σ, τ			

Table 1. Basic logics, axiomatic characterisations, and **L**-accessibility $\rhd$.

Traditionally, the notion of satisfaction relates a world in a model with a formula or a set of formulae. For formulae annotated with ground labels, this notion must be extended by a further "interpretation function" mapping ground labels to worlds; see [7, 11]. If labels contain free variables and, in particular, universal variables, then this notion must also cover all possible instantiations of the universal variables, thus catering for many different "interpretation functions". We extend the notion of satisfiability so it is naturally preserved by our tableau expansion rules, and so that a "closed tableau" is not satisfiable.

We proceed incrementally by defining satisfiability for: ground labels; ground tableau formulae; non-ground tableau formulae; and finally for whole tableaux. But first we enrich models by the "interpretation function" that maps labels to worlds. Note that such interpretations give a meaning to *all* ground labels, not just to those that appear in a particular tableau.

Definition 11. An **L**-interpretation is a pair $\langle \mathbf{M}, \mathbf{I} \rangle$, where $\mathbf{M} = \langle W, R, V \rangle$, is an **L**-model and $\mathbf{I} : \mathcal{L} \to W \cup \{\bot\}$ is a function interpreting ground labels such that: (a) $\mathbf{I}(1) \in W$; (b) $\mathbf{I}(\sigma.(n)) = \mathbf{I}(\sigma.n)$ for all $\sigma.n$ and $\sigma.(n)$ in $\mathcal{L}$; (c) for all $\sigma \in \mathcal{L}$, if $\mathbf{I}(\tau) = \bot$ for some $\tau \in \mathrm{ipr}(\sigma)$ then $\mathbf{I}(\sigma) = \bot$; (d) if $\sigma \rhd \tau$, $\mathbf{I}(\sigma) \in W$, $\mathbf{I}(\tau) \in W$, and $\mathbf{I}(\sigma)$ is idealisable, then $\mathbf{I}(\sigma)\, R\, \mathbf{I}(\tau)$.

Definition 12. An **L**-interpretation $\langle \mathbf{M}, \mathbf{I} \rangle$, where $\mathbf{M} = \langle W, R, V \rangle$, **satisfies** a ground label σ, if for all labels $\tau.n \in \mathrm{ipr}(\sigma) \cup \{\sigma\}$ (that end in an unconditional label position): $\mathbf{I}(\tau) \in W$ implies $\mathbf{I}(\tau.n) \in W$. The **L**-interpretation $\langle \mathbf{M}, \mathbf{I} \rangle$ **satisfies** a ground tableau formula $X : \Delta : \sigma : A$, if (a) $\mathbf{I}(\sigma) = \bot$, or $\mathbf{I}(\tau) = \bot$ for some $\tau \in \Delta$, or $\mathbf{I}(\sigma) \models A$; and (b) if $\mathbf{I}(\tau) \in W$ for all $\tau \in \Delta$, then $\langle \mathbf{M}, \mathbf{I} \rangle$ satisfies σ.

Thus, a tableau formula is satisfied by default if its label σ is undefined (that is, if $\mathbf{I}(\sigma) = \bot$) or if one of the labels in Δ is undefined. But because we deal only with strongly generated sets of labels with root 1, Definitions 11 and 12 force $\mathbf{I}$ to "define" as many members of $\mathrm{ipr}(\sigma)$ as is possible. However, for a conditional ground label of the form $\tau.(n)$, where n is parenthesised, it is perfectly acceptable to have $\mathbf{I}(\tau.(n)) = \bot$ even if $\mathbf{I}(\tau) \in W$.

Example 1. If $\langle \mathbf{M}, \mathbf{I} \rangle$ satisfies $\sigma = 1.1.1$, then $\mathbf{I}(1)$, $\mathbf{I}(1.1)$, and $\mathbf{I}(1.1.1)$ must be defined. If $\sigma = 1.(1).1$, then $\mathbf{I}(1.(1))$ need not be defined; but if it is, then $\mathbf{I}(1.(1).1)$ must be defined.

The domain of every interpretation function $\mathbf{I}$ is the set of all *ground* labels $\mathcal{L}$, but our tableaux contain labels with variables. We therefore introduce a definition of satisfiability for non-ground tableau formulae capturing our intuitions that a label $\sigma.(x)$ stands for *all* possible successors of the label σ, and taking into account the special nature of universal variables.

Definition 13. Given an $\mathbf{L}$-interpretation $\langle \mathbf{M}, \mathbf{I} \rangle$ and a grounding substitution μ, a (non-ground) tableau formula $\phi = X : \Delta : \sigma : A$ is **satisfied** by $\langle \mathbf{M}, \mathbf{I}, \mu \rangle$, written as $\langle \mathbf{M}, \mathbf{I}, \mu \rangle \models \phi$, if, for all grounding substitutions λ, the ground formula $\phi\lambda_{|X}\mu$ is satisfied by $\langle \mathbf{M}, \mathbf{I} \rangle$ (Def. 12). A set $\mathcal{F}$ of tableau formulae is satisfied by $\langle \mathbf{M}, \mathbf{I}, \mu \rangle$, if every member of $\mathcal{F}$ is simultaneously satisfied by $\langle \mathbf{M}, \mathbf{I}, \mu \rangle$.

In the above definition, a ground formula $\phi\lambda_{|X}\mu$ is constructed from ϕ in two steps such that the definition of satisfiability for ground formulae can be applied. To cater for the differences between the free variables and universal variables, we use two substitutions: a fixed substitution μ and an arbitrary substitution λ. The first step, applying $\lambda_{|X}$ to ϕ, instantiates the universal variables $x \in X$. The second step, applying μ to $\phi\lambda_{|X}$, instantiates the free variables. Therefore, the instantiation of universal variables $x \in X$ is given by the arbitrary substitution λ, and the instantiation of free variables $x \notin X$ is given by the fixed substitution μ.

Note, that in the following definition of satisfiable tableaux, there has to be a single satisfying $\mathbf{L}$-interpretation for *all* grounding substitutions μ.

Definition 14. A tableau $\mathcal{T}$ is $\mathbf{L}$-satisfiable if there is an $\mathbf{L}$-interpretation $\langle \mathbf{M}, \mathbf{I} \rangle$ such that for *every* grounding substitution μ there is some *open* branch $\mathcal{B}$ in $\mathcal{T}$ with $\langle \mathbf{M}, \mathbf{I}, \mu \rangle \models \mathcal{B}$.

4 The Calculus

We now present an overview of our calculus, highlighting its main principles.

Our calculus is a refutation method. That is, to prove that a formula A is a theorem of logic $\mathbf{L}$, we first convert its negation $\neg A$ into NNF obtaining a formula B, and then test if B is $\mathbf{L}$-unsatisfiable. To do so, we start with the initial tableau whose single node is $\emptyset : \emptyset : 1 : B$ and repeatedly apply the tableau expansion rules, the substitution rule, and the closure rule until a closed tableau has

been constructed. Since our rules preserve **L**-satisfiability of tableaux, a closed tableau indicates that B is indeed **L**-unsatisfiable, and hence that its negation A is **L**-valid. Since **L**-frames characterise the logic **L** we then know that A is a theorem of logic **L**. Constructing a tableau for $\emptyset : \emptyset : 1 : B$ can be seen as a search for an **L**-model for B. Each branch is a partial definition of a possible **L**-model, and different substitutions give different **L**-models. Our tableau rules extend *one* particular branch using *one* particular formula, thus differing *crucially* from the systematic methods in [6, 11] where a rule extends *all* branches that pass through one particular formula.

Free variables are used in the labels so that when the box-rule is applied in a world, the actual *ground* label of the successor world does not have to be guessed. Instead, free variables can be instantiated immediatedly before a branch is closed to make that closure possible. Note, however, that one single instantiation of the free variables has to be found that allows us to close all branches of a tableau *simultaneously*, and that instantiating a free variable (in the wrong way) to close one branch, can make it impossible to close other branches.

Because a world may have no successor, variable positions in labels have to be conditional to preserve soundness for non-serial logics.

Every variable is introduced into a label by the reduction of a box-formula like $\Box A$. Such a variable x in a tableau formula ϕ on branch $\mathcal{B}$ is "universal" if a renaming $\phi' = \phi\{x := x'\}$ of ϕ could be added to $\mathcal{B}$ without generating additional branches. That is, the modified tableau would be no more difficult to close than the original. An easy way to generate the renaming is to repeat the rule applications that lead to the generation of ϕ, starting from the box-rule application that created x. Once the renaming ϕ' is present on $\mathcal{B}$, the variable x never has to be instantiated to close $\mathcal{B}$ because ϕ' could be used instead of ϕ, thus instantiating x' instead of x. However, if x occurs on two separate branches in the tableau, then x is not universal because repeating these rule applications would generate at least one additional branch. Since the only rule that causes branching is the disjunctive rule, the two separate occurrences of x must have been created by a disjunctive rule application. Therefore, an application of the disjunctive rule to a formula ψ causes the universal variables of ψ to become free variables. Thus, all free variables are a result of a disjunction within the scope of a $\Box$, corresponding to the fact that $\Box$ does not distribute over $\vee$.

When the disjunctive rule "frees" universal variables, additional copies of the box-formula that generated them are needed. However, these additional copies are not generated by the box-rule, but by the disjunctive rule itself.

Our diamond-rule does not introduce a *new* label $\sigma.n$, when it is applied to $X : \Delta : \sigma : \Diamond A$. Instead, each formula $\Diamond A$ is assigned its own unique label $\lceil A \rceil$ which is a Gödelisation of A itself. This rule is easier to implement than the traditional one; and it guarantees that the number of different labels (of a certain length) in a proof is finite, thus restricting the search space.

The box-rule for symmetric and euclidean logics can shorten labels. For example, the tableau formula $X' : \Delta' : 1 : A$ is derived from $X : \Delta : 1.(1) : \Box A$ if the logic is symmetric. The semantics for serial logics guarantee that all labels define

worlds, but in non-serial logics, the label 1 may be defined even though 1.(1) is undefined. To ensure that the formula $X' : \Delta' : 1 : A$ or one of its descendants is used to close a branch only if the label 1.(1) *is* defined, the label 1.(1) is made part of Δ' (see Section 4.2). Such problems do not occur when rule applications always lengthen labels since τ has to be defined if $\tau.l$ is defined.

All expansion rules are sound and *invertible* (some denominator of each rule is **L**-satisfiable *iff* the numerator is **L**-satisfiable). Thus, unlike traditional modal tableau methods where the order of (their non-invertible) rule applications is crucial [6, 11], the order of rule application is *immaterial*.

The differences in the calculi for different logics **L** is mainly in the box-rule, with different denominators for different logics. In addition, a simpler version of the closure rule can be used if the logic is serial.

4.1 Tableau Expansion Rules

There are four expansion rules, one for each type of complex (non-literal) formula. If we wanted to avoid NNF we would have four formula classes (α, β, ν, π) a la Smullyan [6], and an extra rule for double negation. Since we assume that all our formulae are in NNF, we need just one representative for each of the four classes.

As usual, in each rule, the formula above the horizontal line is its *numerator* (the premiss) and the formula(e) below the horizontal line, possibly separated by vertical bars, are its *denominators* (the conclusions). All expansion rules (including the box-rule) are "destructive"; that is, once the (appropriate) rule has been applied to a formula occurrence to expand a branch, that formula occurrence is not used again to expand that branch. Note that we permit multiple occurrences of the same formula on the same branch.

Definition 15. Given a tableau $\mathcal{T}$, a new tableau $\mathcal{T}'$ may be constructed from $\mathcal{T}$ by applying one of the **L-expansion rules** from Table 2 as follows: If the numerator of a rule occurs on a branch $\mathcal{B}$ in $\mathcal{T}$, then the branch $\mathcal{B}$ is extended by the addition of the denominators of that rule. For the disjunctive rule the branch splits and the formulae in the right and left denominator, respectively, are added to the two resulting sub-branches instead.

The box-rule(s) shown in Table 2 require explanation. The form of the rule is determined by the index **L** in the accompanying table. But some of the denominators have side conditions that determine when they are applicable. For example, the constraint $\sigma_6 = 1.l_6$ means that (5) is part of the denominator only when the numerator of the box-rule is of the form $X : \Delta : 1.l_6 : \Box A$. Similarly, the constraints $\sigma_3 = \tau_3.l_3$ and $\sigma_5 = \tau_5.l_5$ for the (4^r) and (B) denominators mean these rules can be used only for a numerator of the form $X : \Delta : \sigma : \Box A$ where $|\sigma| \geq 2$, thereby guaranteeing that the *strictly shorter* labels τ_3 and τ_5 that appear in the respective denominators are properly defined. Note that the (4^d) denominator is the restriction of the (4) denominator to the case where $|\sigma| \geq 2$. The table indicates that the rules for a logic **L** and its serial version **LD** are identical because

$$\frac{X : \Delta : \sigma : A \wedge B}{\begin{array}{c} X : \Delta : \sigma : A \\ X' : \Delta' : \sigma' : B \end{array}}$$

Conjunctive rule. $X' : \Delta' : \sigma' : B$ is a $\mathcal{T}$-renaming of $X : \Delta : \sigma : B$.

$$\frac{X : \Delta : \sigma : A \vee B}{\begin{array}{c|c} \emptyset : \Delta_1 : \sigma_1 : A & \emptyset : \Delta_1 : \sigma_1 : B \\ X_2 : \Delta_2 : \sigma_2 : A \vee B & X_3 : \Delta_3 : \sigma_3 : A \vee B \end{array}}$$

Disjunctive rule. $\psi_i = X_i : \Delta_i : \sigma_i : A \vee B$ are $\mathcal{T}$-renamings of $\psi = X : \Delta : \sigma : A \vee B$ for $1 \leq i \leq 3$ (where the X_i are pairwise disjoint). If $X = \emptyset$ then ψ_2, ψ_3 are omitted.

$$\frac{X : \Delta : \sigma : \Diamond A}{X : \Delta : \sigma.\lceil A \rceil : A}$$

Diamond-rule. $\lceil \cdot \rceil$ is an arbitrary but fixed bijection from the set of formulae to $\mathbf{N}$.

$$\frac{X : \Delta : \sigma : \Box A}{\begin{array}{cc} X \cup \{x\} : \Delta : \sigma.(x) : A & \text{(K)} \\ X_1 \cup \{x_1\} : \Delta_1 : \sigma_1.(x_1) : \Box A & (4) \\ X_2 \cup \{x_2\} : \Delta_2 : \sigma_2.(x_2) : \Box A & (4^d) \\ X_3 : \Delta_3 \cup \{\sigma_3\} : \tau_3 : \Box A & (4^r) \\ X_4 : \Delta_4 : \sigma_4 : A & \text{(T)} \\ X_5 : \Delta_5 \cup \{\sigma_5\} : \tau_5 : A & \text{(B)} \\ X_6 : \Delta_6 \cup \{\sigma_6\} : 1 : \Box\Box A & (5) \end{array}}$$

Box-rule. For $1 \leq i \leq 6$, $X_i : \Delta_i : \sigma_i : \Box A$ are $\mathcal{T}$-renamings of $X : \Delta : \sigma : \Box A$. The variables $x, x_1, x_2 \in \text{Vars}$ are new to $\mathcal{T}$. The sets $X \cup \{x\}$, $X_1 \cup \{x_1\}$, $X_2 \cup \{x_2\}$, X_3, X_4, X_5, and X_6 are pairwise disjoint. In addition, $\sigma_3 = \tau_3.l_3$, $\sigma_5 = \tau_5.l_5$, $\sigma_6 = 1.l_6$, and $|\sigma_2| \geq 2$. The form of the denominator depends on the logic $\mathbf{L}$, and is determined by including every denominator corresponding to the entry for $\mathbf{L}$ in the table below.

Logics	Box-rule denominator	Logics	Box-rule denominator
K, **D**	(K)	**K45**, **K45D**	(K), (4), (4^r)
T	(K), (T)	**K4B**,	(K), (B), (4), (4^r)
KB, **KDB**	(K), (B)	**B**	(K), (T), (B)
K4, **KD4**	(K), (4)	**S4**	(K), (T), (4)
K5, **KD5**	(K), (4^d), (4^r), (5)	**S5**	(K), (T), (4), (4^r)

Table 2. Tableau expansion rules.

these logics are distinguished by the form of our closure rule; see Definition 18. Various other ways to define the calculi for serial logics exist; see [11].

4.2 The Substitution Rule and the Closure Rule

By definition, the substitution rule allows us to apply *any* substitution at *any* time to a tableau. In practice, however, it makes sense to apply only "useful" substitutions; that is, those most general substitutions which allow to close a branch of the tableau.

Definition 16. Substitution rule: Given a tableau $\mathcal{T}$, a new tableau $\mathcal{T}' = \mathcal{T}\sigma$ may be constructed from $\mathcal{T}$ by applying a substitution σ to $\mathcal{T}$ that instantiates free variables in $\mathcal{T}$ with other free variables or natural numbers.

In tableaux for modal logics without free variables as well as in free-variable tableaux for first-order logic, a tableau branch is closed if it contains comple-

mentary literals since this immediately implies the existence of an inconsistency. Here, however, this is not always the case because the labels of the complementary literals may be conditional. For example, the (apparently contradictory) pair $\emptyset : 1.(1) : p$ and $\emptyset : 1.(1) : \neg p$ is not necessarily inconsistent since the world $\mathbf{I}(1.(1))$ may not exist in the chosen model. Before declaring this pair to be inconsistent, we therefore have to ensure that $\mathbf{I}(1.(1)) \neq \bot$ for all $\mathbf{L}$-interpretations satisfying the tableau branch $\mathcal{B}$ that is to be closed. Fortunately, this knowledge can be deduced from other formulae on $\mathcal{B}$. Thus in our example, a formula like $\psi = X : 1.1 : A$ on $\mathcal{B}$ would "justify" the use of the literal pair $\emptyset : 1.(1) : p$ and $\emptyset : 1.(1) : \neg p$ for closing the branch $\mathcal{B}$ since any $\mathbf{L}$-interpretation $\langle \mathbf{M}, \mathbf{I} \rangle$ satisfying $\mathcal{B}$ has to satisfy ψ, and, thus, $\mathbf{I}(1.(1)) = \mathbf{I}(1.1) \neq \bot$ has to be a world in the chosen model $\mathbf{M}$. The crucial point is that the label 1.1 of ψ is *unconditional* exactly in the *conditional* positions of $\emptyset : 1.(1) : p$ and $\emptyset : 1.(1) : \neg p$. These observations are now extended to the general case of arbitrary *ground* labels.

Definition 17. A ground label σ with j-th position $[n_j]$ $(1 \leq j \leq |\sigma|)$ is **justified** on a branch $\mathcal{B}$ if there is some set $\mathcal{F} \subseteq \mathcal{B}$ of tableau formulae such that for every j: (a) some label in $\mathrm{lab}(\mathcal{F})$ has (an unconditional but otherwise identical) j-th position n_j; and (b) for all $\tau \in \mathrm{lab}(\mathcal{F})$: if $|\tau| \geq j$ then the j-th position in τ is n_j or (n_j).

Definition 18. Given a tableau $\mathcal{T}$ and a substitution $\rho : \mathrm{Univ}(\mathcal{T}) \to \mathbf{N}$ that instantiates universal variables in $\mathcal{T}$ with natural numbers, the **L-closure rule** allows to construct a new tableau $\mathcal{T}'$ from $\mathcal{T}$ by marking $\mathcal{B}$ in $\mathcal{T}$ as closed provided that: (a) the branch $\mathcal{B}\rho$ of $\mathcal{T}\rho$ contains a pair $X : \Delta : \sigma : p$ and $X' : \Delta' : \sigma : \neg p$ of complementary literals; and (b1) the logic $\mathbf{L}$ is serial, or (b2) all labels in $\{\sigma\} \cup \Delta \cup \Delta'$ are ground and justified on $\mathcal{B}\rho$.

Note that the substitution ρ that instantiates universal variables is not actually applied to the tableau when the branch is closed; it only has to exist.

By definition, only complementary *literals* close tableau branches, but in theory, pairs of complementary *complex formulae* could be used as well.

4.3 Tableau Proofs

We now have all the ingredients we need to define the notion of a tableau proof.

Definition 19. A sequence $\mathcal{T}^0, \ldots, \mathcal{T}^r$ of tableaux is an **L-proof** for the $\mathbf{L}$-unsatisfiability of a formula A if: (a) $\mathcal{T}^0$ consists of the single node $\emptyset : \emptyset : 1 : A$; (b) for $1 \leq m \leq r$, the tableau $\mathcal{T}^m$ is constructed from $\mathcal{T}^{m-1}$ by applying an $\mathbf{L}$-expansion rule (Def. 15), the substitution rule (Def. 16), or the $\mathbf{L}$-closure rule (Def. 18); and (c) all branches in $\mathcal{T}^r$ are marked as closed.

Theorems 20 and 22 state soundness and completeness for our calculus with respect to the Kripke semantics for logic $\mathbf{L}$; the proofs can be found in [1].

Theorem 20 (Soundness). *Let A be a formula in NNF. If there is an $\mathbf{L}$-proof $\mathcal{T}^0, \ldots, \mathcal{T}^r$ for the $\mathbf{L}$-unsatisfiability of A (Def. 19), then A is $\mathbf{L}$-unsatisfiable.*

We prove completeness for the non-deterministic and unrestricted version of the calculus, and also for all tableau procedures based on this calculus that deterministically choose the next formula for expansion (in a *fair* way) and that only apply most general closing substitutions.

Definition 21. Given an open tableau $\mathcal{T}$, a *tableau procedure* Ψ deterministically chooses an open branch $\mathcal{B}$ in $\mathcal{T}$ and a non-literal tableau formula ψ on $\mathcal{B}$ for expansion.

The tableau procedure Ψ is fair if, in the (possibly infinite) tableau that is constructed using Ψ (where no substitution is applied and no branch is closed), every formula has been used for expansion of every branch on which it occurs.

Theorem 22 (Completeness). *Let Ψ be a fair tableau procedure, and let A be an* **L***-unsatisfiable formula in NNF. Then there is a (finite) tableau proof $\mathcal{T}^0, \ldots, \mathcal{T}^r$ for the* **L***-unsatisfiability of A, where $\mathcal{T}^i$ is constructed from $\mathcal{T}^{i-1}$ ($1 \leq i \leq r$) by (a) applying the appropriate* **L***-expansion rule to the branch $\mathcal{B}$ and the formula ψ on $\mathcal{B}$ chosen by Ψ from $\mathcal{T}^{i-1}$; or (b) applying a most general substitution such that the* **L***-closure rule can be applied to a previously open branch in $\mathcal{T}^{i-1}$.*

Example 2. We prove that $A = \Box(p \to q) \to (\Box p \to (\Box q \wedge \Box p))$ is a **K**-theorem. To do this, we first transform the negation of A into NNF; the result is $B = \mathrm{NNF}(\neg A) = \Box(\neg p \vee q) \wedge \Box p \wedge (\Diamond \neg q \vee \Diamond \neg p)$. The (fully expanded) tableau $\mathcal{T}$, that is part of the proof for the **K**-unsatisfiability of B is shown in Figure 1. The nodes of the tableau are numbered; a pair $[i; j]$ is attached to the i-th node, the number j denotes that node i has been created by applying an expansion rule to the formula in node j. Note, that by applying the disjunctive rule to 6, the nodes 11 to 14 are added; 13 and 14 are renamings of 6. The variable y_1 is no longer universal in 11 and 12.

When the substitution $\sigma = \{y_1/\lceil \neg q \rceil\}$ is applied to $\mathcal{T}$, the branches of the resulting tableau $\mathcal{T}\sigma$ can be closed as follows, thereby completing the tableau proof: The left branch $\mathcal{B}_1$ of $\mathcal{T}\sigma$ can be closed by the universal variable substitution $\rho_1 = \{x/\lceil \neg q \rceil\}$ because $\mathcal{B}_1\rho_1$ then contains the complementary pair $\{\lceil \neg q \rceil\} : 1.(\lceil \neg q \rceil) : p$ and $\emptyset : 1.(\lceil \neg q \rceil) : \neg p$ in nodes 7 and 11, respectively. The label $1.(\lceil \neg q \rceil)$ of these literals is justified on $\mathcal{B}_1\rho_1$ by label $1.\lceil \neg q \rceil$ of formula 10. In this case, the complementary literals contain conditional labels which are only justified by a third formula on the branch, so checking for justification is indispensable. The middle branch $\mathcal{B}_2$ of $\mathcal{T}\sigma$ can be closed using the same universal variable substitution $\rho_2 = \rho_1 = \{x/\lceil \neg q \rceil\}$ as for the left branch. The branch $\mathcal{B}_2\rho_2$ then contains the complementary literals $\{\lceil \neg q \rceil\} : 1.(\lceil \neg q \rceil) : q$ and $\emptyset : 1.(\lceil \neg q \rceil) : \neg q$ in nodes 10 and 12. The label is again justified by formula 10, which in this case is one of the complementary literals. Note that the middle branch in $\mathcal{T}$ can be closed only by the substitution $\sigma = \{y_1/\lceil \neg q \rceil\}$, other choices will not suffice. The right branch $\mathcal{B}_3$ of $\mathcal{T}\sigma$ can be closed using the universal variable substitution $\rho_3 = \{x/\lceil \neg p \rceil\}$ as $\mathcal{B}_3\rho_3$ then contains the pair $\{\lceil \neg p \rceil\} : 1.(\lceil \neg p \rceil) : p$ and $\{\lceil \neg p \rceil\} : 1.\lceil \neg p \rceil : \neg p$ of complementary literals in nodes 7 resp. 15. The label $1.(\lceil \neg p \rceil)$ of node 7 is justified on $\mathcal{B}_3$ by formula 15.

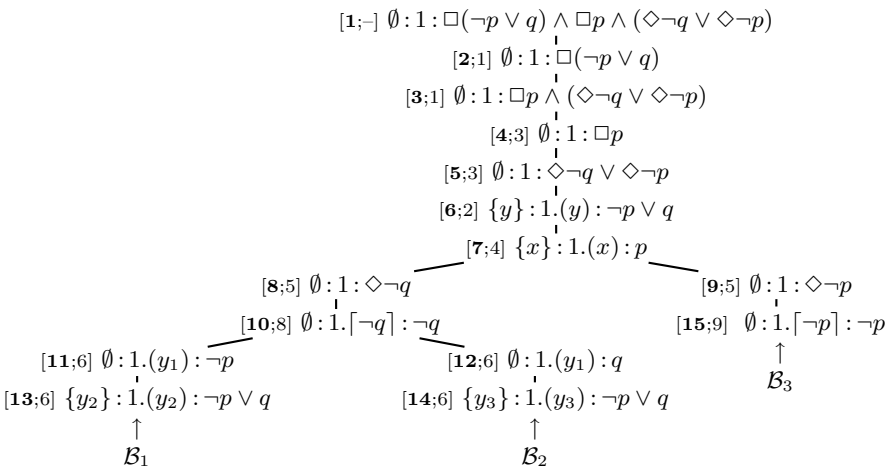

Fig. 1. The tableau $\mathcal{T}$ from Example 2.

The universal variable substitution $\rho_1 = \rho_2 = \{x/\lceil\neg q\rceil\}$ that closes $\mathcal{B}_1$ and $\mathcal{B}_2$ is incompatible with the substitution $\rho_3 = \{x/\lceil\neg p\rceil$ that closes $\mathcal{B}_3$. Therefore, if the variable x were not universal in formula 7, the tableau could not be closed; a second instance of formula 7 would have to be added (which in this example would not do much harm).

5 leanK: An Implementation

We have implemented our calculus as a "lean" theorem prover written in Prolog (the source code is available at `http://i12www.ira.uka.de/modlean` on the *World Wide Web*). The basic version leanK, for the logic **K**, consists of just eleven Prolog clauses and 45 lines of code. The version for the logic **KD**, which allows unjustified labels, is even shorter: it consists of only 6 clauses and 27 lines of code.

The leanK program employs the following fair tableau procedure: Given a tableau $\mathcal{T}$, the branch that is expanded next is the left-most open branch, with the formulae on any particular branch organised as a queue. The first formula in the chosen branch/queue is removed from the queue and is used to update the tableau as follows: If the chosen formula is not a literal then some (one) rule is applicable to it, and the formulae created by that rule application are added to the queue as follows: if the (traditional part of the) created formula is strictly less complex than the numerator, this new formula is added to the front of the queue, otherwise it is added to the end of the queue. In particular, this means that renamings of formulae added by the disjunctive rule, and the formula labelled (4) and (4^r) in the denominator of the box-rule, are added to

the end of the queue. If the chosen formula in the queue is a literal ϕ and there is a most general substitution μ of the free variables in ϕ such that $\phi\mu$ and some other literal $\psi\mu$ on the branch can be used for closure, then there is a choice point: (1) the substitution μ may be applied and the branch closed, or (2) the literal is removed from the queue and the next formula moves to the front. There is a further choicepoint if there is more than one closing substitution μ. In case no closing substitution μ exists, option (2) is used deterministically. If there is a choice, Prolog's backtracking mechanism is used to resolve this non-determinism and explore all choices.

Limiting the number of free variables in a branch forces every branch to terminate after some finite number of rule applications. Prolog's backtracking mechanism then automatically processes the next branch in the queue. Iterative deepening preserves completeness by increasing this branch limit, step by step, as long as no proof can be found.

A lean and efficient implementation is only possible by making use of Prolog's special features: backtracking is used to resolve the non-determinism in the tableau procedure; built-in unification is used for finding most general closing substitutions and for the justification test; and first-argument clause indexing is employed to quickly determine the appropriate tableau rule for the next formula.

To avoid generating useless renamings of disjunctive formulae, the version of leanK used to obtain statistics uses the following restriction: when the disjunctive rule is applied to a formula $\psi = X : \Delta : \sigma : A \vee B$, the renaming ψ_2 (resp. ψ_3) created by the disjunctive rule is "protected" from further applications of the disjuntive rule until one of the variables in X_1 has been instantiated. That is, a renaming is useful only when one of the variables in X_1 has been used to close a branch using (a descendant of) $\emptyset : \Delta_1 : \sigma_1 : A$ or $\emptyset : \Delta_1 : \sigma_1 : B$.

The following table shows statistics for a set of 72 **K**-theorems kindly provided by A. Heuerding. Of these, leanK could prove 61 in the allotted time of 15sec, with 52 in less than 10msec (not shown in the table). The program was terminated if no proof had been found after 15sec. The table shows the number of branches that were closed, the maximal number of free variables in a branch, and the proof time (running under SICStus Prolog on a SUN Ultra 1 workstation).

No.	24	44	46	50	52	55	56	67	72
Branches	22251	90	137	43	56	1011	68	26565	154
Var.-Limit	10	5	5	4	4	11	4	11	6
Time [msec]	4400	50	80	20	30	1000	30	9520	90

The examples that took several seconds to prove show an advantage of lean implementations: the very high inference rate of about 2500 closed branches per second. The complexity of these formulae is non-trivial; one of the more complex ones, No. 55, is:

$$((\Box(\Box(p\to\Box p)\to p)\to p)\wedge(\Box(\Box((\Box(p\to\Box p)\to p)\wedge(\Box(\Box(p\to\Box p)\to p)\to$$
$$\Box\Box(\Box(p\to\Box p)\to p))\to\Box((\Box(p\to\Box p)\to p)\wedge(\Box(\Box(p\to\Box p)\to p)\to\Box\Box(\Box(p\to\Box p)\to p))))\to$$
$$(\Box(p\to\Box p)\to p)\wedge(\Box(\Box(p\to\Box p)\to p)\to\Box\Box(\Box(p\to\Box p)\to p)))\to(\Box(p\to\Box p)\to p)\wedge$$
$$(\Box(\Box(p\to\Box p)\to p)\to\Box\Box(\Box(p\to\Box p)\to p)))\to\Box(\Box(p\to\Box p)\to p)\to p\vee\Box p)$$

6 Conclusion and Future Work

Our initial results, presented in the last section, are very encouraging. We believe that labels with variables deliver the following advantages:

- The use of variables generates a smaller search space since a label can now stand in for all its ground instances. This is in stark contrast to the modular systems of [17], where only ground labels are used.
- The use of a Gödelisation function in the diamond-rule leads to a smaller number of labels than in other labelled tableau methods since two different occurrences of the formula $X : \Delta : \sigma : \Diamond A$ lead to the same formula $X : \Delta : \sigma.\lceil A \rceil : A$. We therefore do not need to delete duplicate occurrences of a formula as is done in some tableau implementations for modal logics. This is particularly important since the world $\sigma.\lceil A \rceil$ may be the root of a large sub-model and duplicating it is likely to be extremely inefficient.
- The use of universal variables can exponentially shorten the length of proofs (see [1]), with only a minor increase in the implementation overheads.
- Our "lean" implementation is perfect for applications where the deductive engine must be transparent and easily modifiable.

Our method is really a very clever translation of propositional modal logics into first-order logic, and most of the complications arise because some worlds may have no successors. The new notion of conditional labels allows us to keep track of these complications, and thus handle the non-serial logics that frustrate other "general frameworks" [9, 14]. Nevertheless, our method can also handle *second order* "provability" logics like **G** and **Grz**; see [11]. Furthermore, specialised versions of these tableau systems can match the theoretical lower bounds for particular logics like **K45**, **G** and **Grz** if we give up modularity; see [11, 17]. We intend to extend our initial implementation of leanK along these lines.

The 15 basic modal logics are known to be decidable and techniques from [6, 11, 17, 13] can be used to extend our method into a decision procedure. However, it is not clear that this is possible in a *lean* way. The extra implementation restriction mentioned in the previous section is of vital importance here since it is essentially a demand driven *contraction rule* on box-formulae since box-formulae get copied only as the required free variables get instantiated. And controlling contraction is often the key to decidability.

Fitting [8] has recently shown how to view the original leanTAP program for classical propositional logic as an unusual sequent calculus `dirseq`. He has also shown how to extend `dirseq` to handle the modal logics **K**, **KT**, **K4**, and **S4**. As with traditional modal tableaux, however, `dirseq` does not handle the symmetric logics like **S5** and **B**. We are currently extending our work to give a modular free variable version of `dirseq` that does handle these logics.

It is also possible to extend our method to deal with the notions of global and local logical consequence [6].

An alternative variable label approach [12] uses different unification algorithms to find complementary literals for branch closure. However, the interactions between modalities, variable labels, and unification algorithms is by no

means easy to disentangle. Extending our method to utilise special unification algorithms is perfectly possible, now that correctness and completeness have been worked out for the interactions between modalities and variable labels alone.

References

1. B. Beckert and R. Goré. Free variable tableaux for propositional modal logics. Interner Bericht 41/96, Universität Karlsruhe, Fakultät für Informatik, 1996.
2. B. Beckert, R. Hähnle, P. Oel, and M. Sulzmann. The tableau-based theorem prover $_3T^AP$, version 4.0. In *Proc. CADE-13*, LNCS 1104. Springer, 1996.
3. B. Beckert and J. Posegga. leanTAP: Lean tableau-based deduction. *Journal of Automated Reasoning*, 15(3):339–358, 1995.
4. E. Bencivenga. Free logic. In D. Gabbay and F. Günthner, editors, *Handbook of Philosophical Logic*, volume 3. Kluwer, Dordrecht, 1986.
5. M. D'Agostino, D. Gabbay, and A. Russo. Grafting modalities onto substructural implication systems. *Studia Logica*, 1996. To appear.
6. M. Fitting. *Proof Methods for Modal and Intuitionistic Logics*, volume 169 of *Synthese Library*. D. Reidel, Dordrecht, Holland, 1983.
7. M. Fitting. *First-Order Logic and Automated Theorem Proving*. Springer, second edition, 1996.
8. M. Fitting. Leantap revisited. Draft Manuscript, Jan. 1996.
9. A. Frisch and R. Scherl. A general framework for modal deduction. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proc. 2nd Conference on Principles of Knowledge Representation and Reasoning*. Morgan-Kaufmann, 1991.
10. D. Gabbay. *Labelled Deductive Systems*. Oxford University Press, 1996.
11. R. Goré. Tableau methods for modal and temporal logics. In M. D'Agostino, D. Gabbay, R. Hähnle, and J. Posegga, editors, *Handbook of Tableau Methods*, chapter 7. Kluwer, Dordrecht, 1997. To appear.
12. G. Governatori. A reduplication and loop checking free proof system for S4. In *Short Papers: TABLEAUX'96*, number 154-96 in RI-DSI, Via Comelico 39, 20135 Milan, Italy, 1996. Department of Computer Science, University of Milan.
13. A. Heuerding, M. Seyfried, and H. Zimmermann. Efficient loop-check for backward proof search in some non-classical logics. In P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, editors, *Proc. TABLEAUX'96*, LNCS 1071. Springer, 1996.
14. P. Jackson and H. Reichgelt. A general proof method for first-order modal logic. In *9th Int. Joint Conference on Artificial Intelligence*, pages 942–944, 1987.
15. S. Kanger. *Provability in Logic*. Stockholm Studies in Philosophy, University of Stockholm. Almqvist and Wiksell, Sweden, 1957.
16. R. Letz, J. Schumann, S. Bayerl, and W. Bibel. SETHEO: A high-performance theorem prover. *Journal of Automated Reasoning*, 8(2):183–212, 1992.
17. F. Massacci. Strongly analytic tableaux for normal modal logics. In A. Bundy, editor, *Proc., CADE-12*, LNCS 814. Springer, 1994.
18. G. Mints. *A Short Introduction to Modal Logic*. CSLI, Stanford, 1992.
19. S. Reeves. Semantic tableaux as a framework for automated theorem-proving. In C. Mellish and J. Hallam, editors, *Advances in Artificial Intelligence*. Wiley, 1987.
20. A. Russo. Generalising propositional modal logic using labelled deductive systems. In F. Baader and K. Schulz, editors, *Proceedings FroCoS*. Kluwer, 1996.

This article was processed using the LaTeX macro package with LLNCS style