System Description: leanK 2.0

Bernhard Beckert[1] Rajeev Goré[2,⋆]

[1] University of Karlsruhe, Institute for Logic, Complexity and Deduction Systems, D-76128 Karlsruhe, Germany. E-mail: beckert@ira.uka.de
[2] Automated Reasoning Project and Department of Computer Science, Australian National University, Canberra, ACT, 0200, Australia. E-mail: rpg@arp.anu.edu.au

Abstract. leanK is a "lean", i.e., extremely compact, Prolog implementation of a free variable tableau calculus for propositional modal logics. leanK 2.0 includes additional search space restrictions and fairness strategies, giving a decision procedure for the logics **K**, **KT**, and **S4**.

Overview. leanK is a "lean" Prolog implementation of the free variable tableau calculus for propositional modal logics reported in [1]. It performs depth first search and is based upon leanT^AP [2]. Formulae are annotated with labels containing variables, which capture the universal and existential nature of the box and diamond modalities, respectively. leanK 2.0 includes additional search space restrictions and fairness strategies, giving a decision procedure for the logics **K**, **KT**, and **S4**. It has 87, 51, and 132 lines of code for **K**, **KD**, and **S4**, respectively.

The main advantages of leanK are its modularity and its versatility. Due to its small size, leanK is easier to understand than a complex prover, and hence easier to adapt to special needs. Minimal changes in the rules give provers for all the 15 basic normal modal logics. By sacrificing modularity we can obtain specialised (faster) provers for particular logics like **K45D**, **G** and **Grz**. It is easy to obtain an explicit counter-example from a failed proof attempt. The leanK (2.0) SICStus Prolog 3 code is at: `http://i12www.ira.uka.de/modlean`.

The Calculus. We describe leanK (2.0) in some detail since [1] does not contain the new search space restrictions and fairness strategies.

To reduce the number of tableau rules, we assume all formulae are in negation normal form (NNF). An NNF transformation comes with the leanK source code.

To test a formula A for theoremhood in logic **L**, leanK tests whether the formula $B = \mathrm{NNF}(\neg A)$ is **L**-unsatisfiable. The initial (single node) tableau contains the labelled formula $1 : B$. leanK repeatedly applies the tableau expansion and closure rules until (a) a closed tableau is constructed (whence B is unsatisfiable and A is a theorem) or (b) no further rule applications are possible (whence B is **L**-satisfiable and A is not a theorem). The following features distinguish leanK's calculus from other labelled modal tableau calculi (see [4] for an overview):

⋆ Supported by the Australian Research Council via a Queen Elizabeth II Fellowship.

Free variables in labels: Applying the traditional 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 successors, variable positions in labels can be "conditional" (i.e., a formula labelled with a conditional label σ has only to be satisfied by a model if the world corresponding to σ exists in that model). Similar ideas have been explored in [6] and [5] using unification of labels, rather than just matching (as in our calculus), and also using an analytic cut rule.

Universal variables: Under certain conditions, a variable x in a label 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 instead (a Gödelisation of) the formula A itself as the label, 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 intuitive reading of a labelled tableau formula $\sigma : A$ (where σ is a label and A is a modal formula in NNF) is "the possible world σ satisfies the formula A". 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, in a non-serial logic, the possible world 1 may have no successor worlds; so, in such logics, we 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". Our rule for disjunctions retains free variables in the labels of the two disjuncts, but because $\Box$ does not distribute over $\vee$, such variables then lose their "universal" force. These "rigid" variables can be instantiated only *once* in a tableau proof. When the disjunctive rule makes universal variables rigid, additional copies of the box-formula that generated the original variables are needed. However, these additional copies are not generated by the box-rule, but by the disjunctive rule itself. In the formulae resulting from expansion rule applications, universal variables are renamed so each universal variable occurs in only one formula in a tableau.

All of lean**K**'s tableau expansion rules are *invertible*: some denominator (conclusion) of each rule is satisfiable *iff* the numerator (the premiss) is satisfiable. Thus, unlike traditional modal tableau methods [3, 4], the order of rule application is *immaterial.* The rules for expanding a branch can be found in [1].

The calculi for different logics differ mainly in the box-rule, with different denominators for different logics (see [1]). In addition, a simpler version of the closure rule can be used if the logic is serial. Since each rule corresponds to a separate Prolog clause, replacing one clause with another implements a different logic. The clauses for some logics require additional arguments so minor editing is also required to ensure all clauses contain the same number of arguments. If labels contain free variables, detecting closure in non-serial logics is non-trivial because the labels of apparently complementary literals may be conditional. The (apparently contradictory) pair $1.(1):p$ and $1.(1):\neg p$ is not necessarily inconsistent since the world represented by $1.(1)$ may not exist in the chosen model. We therefore have to ensure that this world exists in all interpretations satisfying the tableau branch $\mathcal{B}$, before closing $\mathcal{B}$. This knowledge can be deduced from other formulae on $\mathcal{B}$. Thus in our example, a formula like $1.1:A$ on $\mathcal{B}$ would "justify" the use of the literal pair $1.(1):p$ and $1.(1):\neg p$ for closing the branch $\mathcal{B}$. The crucial point is that the label 1.1 is *unconditional* exactly in the *conditional* position of $1.(1)$. In that case, we say that the label $1.(1)$ is *justified* on the branch $\mathcal{B}$ (for a formal definition see [1]).

The Fair Proof Procedure. The calculus described above is sound. Using a fair proof search procedure it is also complete. lean**K** uses a fairness strategy for closing branches so backtracking over different choices of complementary literals on a tableau branch and the closing substitutions associated with them is unnecessary. For that purpose, each tableau formula ϕ has an attached list of all instantiations of the rigid variables in ϕ that have previously been applied to copies of ϕ occurring on the same branch. Closure on a pair of complementary literals on a branch is forbidden if the associated closing substitution would lead to a previous instantiation of the free variables. Furthermore, to avoid generating useless renamings of disjunctive formulae, lean**K** uses the following restriction: when the disjunctive rule is applied to a formula $\phi = \sigma : A \lor B$, the renamings of ϕ that are added to the new sub-branches are "put asleep". The disjunctive rule is not applied to these renamings until they are woken up, which is only allowed if at least one of the free variables in σ has been instantiated using (a descendant of) $\sigma : A$ (resp. $\sigma : B$) for closure.

The next branch for expansion and the next formula to which a tableau rule is applied are chosen using the following fair procedure: always choose the left-most open branch, and view the formulae on any particular branch as a queue. The first formula in the branch/queue is removed and is used as the numerator to update the tableau as follows (a disjunctive rule is only used if it is not asleep): 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. To preserve fairness, if (the traditional part of) the created formula is more complex than the numerator, this new formula is added to the end of the queue, otherwise it is added to the front of the queue. In particular, renamings of disjunctive formulae added by the disjunctive rule, and the transitive part of the denominator of the box-rule are added to the end. If the queue is empty, the first sleeping disjunctive formula that can be woken up is used; if there are none, the proof search terminates.

This procedure is a semi-decision procedure for all basic modal logics and, due to the finite diamond-rule, a decision procedure for the non-transitive logics.

To ensure that proof search terminates in case the logic is transitive, lean**K** 2.0 employs additional search space restrictions to avoid loops: (1) A box-formula may only be used to expand a branch $\mathcal{B}$ if its label is justified on $\mathcal{B}$. (2) A diamond-formula $\sigma : \Diamond A$ may only be used to expand a branch $\mathcal{B}$ if it is *not* "blocked" by a formula $\sigma' : \Diamond A$ that already has been used to expand $\mathcal{B}$ where $\sigma' \leq \sigma$ (i.e., σ' is an initial prefix of σ) up to instantiation of universal variables—except if $\sigma : \Diamond A$ is unblocked "behind" σ' by a *new* box-formula $\tau : \Box B$ ($\sigma' \leq \tau$ and $\tau \leq \sigma$). A formula $\tau : \Box B$ is *new* if, at the time it is used to expand the branch, there is no formula $\tau' : \Box B$ that already has been used to expand $\mathcal{B}$ (where $\tau' \leq \tau$). Intuitively, a diamond-formula $\Diamond A$ may not be used for expansion in a world w if we have already seen it in a world w' that is a predecessor of w, except if we have seen a new box formula in a world w'' that is on the path from w' to w. Now, if the logic is transitive, the next formula to which a tableau rule is applied is chosen from the branch/queue in the following order: (1) the first formula that is a literal, a conjunctive formula, or a disjunctive formula that is not asleep, (2) the first diamond-formula that is not blocked, (3) the first disjunctive formula that can be woken up, (4) the first box-formula whose label is justified; if none of these choices is possible, the proof search terminates.

Performance. The strength of lean**K** (2.0) clearly is its small size and adaptability and not its performance. The following table shows statistics for a set of 72 **K**-theorems kindly provided by A. Heuerding. These formulae are non-trivial; No. 55, has about 90 logical operators. lean**K** 2.0 could prove 58 of these theorems. The program was terminated if no proof had been found after 15sec. The table shows the number of branches that were closed, and the proof time (running under SICStus Prolog 3 on a SUN Ultra 1 workstation).

	No.	24	44	46	50	52	55	56	67	72
Vers. 1.0	Branches	22251	90	137	43	56	1011	68	26565	154
	Time [msec]	4400	50	80	20	30	1000	30	9520	90
Vers. 2.0	Branches	–	6	46	27	15	5	42	–	–
	Time [msec]	–	20	20	20	10	0	20	–	–

References

1. B. Beckert and R. Goré. Free variable tableaux for propositional modal logics. In *Proceedings, TABLEAUX-97*, LNCS 1227, pages 91–106. Springer, 1997.
2. B. Beckert and J. Posegga. leanT^AP: Lean tableau-based deduction. *Journal of Automated Reasoning*, 15(3):339–358, 1995.
3. M. Fitting. *Proof Methods for Modal and Intuitionistic Logics*, volume 169 of *Synthese Library*. D. Reidel, Dordrecht, Holland, 1983.
4. R. Goré. Tableau methods for modal and temporal logics. In *Handbook of Tableau Methods*, Kluwer, Dordrecht, 1998. To appear.
5. G. Governatori. Labelled tableaux for multi-modal logics. In *Proceedings, TABLEAUX-95*, LNCS 918, pages 79–94. Springer, 1995.
6. J. Pitt and J. Cunningham. Distributed modal theorem proving with KE. In *Proceedings, TABLEAUX-96*, LNAI 1071, pages 160–176. Springer, 1996.