Deciding K using ꓘ

Andrei Voronkov
Department of Computer Science
The University of Manchester
voronkov@cs.man.ac.uk

Abstract

We present a bottom-up decision procedure for the propositional modal logic K based on the inverse method. The procedure is based on the "inverted" version ꓘ of a sequent calculus for K. The first (essentially nonoptimized) implementation of the procedure shows its competitiveness with the best state-of-the-art systems.

Note. The use of symbols like ꓘ is not directly supported by the standard text-processing systems. When citing this paper, we suggest to use the title *"Deciding* K *using K-inverse"* or *simply "Deciding* K *using K"*. In LaTeX the symbol ꓘ can be reproduced by using the commands

```
\usepackage{graphicx}
...
\newcommand{\inverseK}{\rotatebox[]{180}{K}}
```

1 Introduction

Nonclassical (propositional) logics play an increasing role in computer science. They are used in model checking, verification, and knowledge representation. Traditional decision procedures for these logics use semantic tableaux [Baader and Hollunder 1991, Balsiger, Heuerding and Schwendimann 1998], optimized tableaux [Horrocks 1998, Patel-Schneider 1998], SAT-based methods [Giunchiglia and Sebastiani 1996, Giunchiglia, Giunchiglia, Sebastiani and Tacchella 1998, Tacchella 1999], or highly optimized translations into first-order classical logic [Hustadt and Schmidt 1997, Schmidt 1998].

In this paper we present a bottom-up decision procedure for propositional modal logic K based on the inverse method. Our techniques can be easily modified to its multi-modal variant K(m) with m modalities, but we restrict ourselves to K for simplicity. The modal logic K(m) is important for several reasons, mainly because this logic is a syntactic variant of the knowledge representation formalism $\mathcal{ALC}$. Our experiments demonstrate that bottom-up procedures can be competitive with the best state-of-the-art methods for K.

We believe that the technique described in our paper opens an interesting research path in the implementation of decision procedures for other important nonclassical logics, such as temporal and description logics.

This paper is organized as follows. In Section 2 we introduce the modal logic K and discuss traditional proof-search procedures for this logic. In Section 3 we define the logical calculus ꓘ used by the inverse method proof procedures, formulate several properties of this calculus and show how proof-search can be organized based on these properties. In Section 4 we discuss several redundancies in ꓘ. Exploitation of the redundancies results in more efficient proof procedures. In Section 5 we briefly describe the implementation of the inverse method in our system Kꓘ and provide experimental evidences showing competitiveness of Kꓘ with the state-of-the-art systems. In Section 6 we consider related work, mainly related to theorem proving by the inverse method. Finally, in Section 7 we discuss future research directions.

Due to the space restrictions we omit proofs in this paper. The full proofs are included in [Voronkov 2000] available on the Web.

2 Preliminaries

In this paper we consider propositional modal logic K. K is the minimal modal logic, i.e. the set of modal formulas valid in all Kripke models. For the discussion of modal logics and Kripke models see any standard references [e.g. Fitting 1983, Wallen 1990]. In this paper, we will speak about (un)satisfiability of formulas instead of the dual notion of *validity*. It is known that the satisfiability problem for K is PSPACE-complete. A number of interesting properties of the knowledge representation formalism $\mathcal{ALC}$, like concept subsumption, can be represented as statements about (un)satisfiability of formulas in the multimodal version K(m) of K.

There are several characterizations of satisfiability for K.

1. The characterization of satisfiability based on Kripke structures gives rise to two kinds of methods: SAT-based methods, in which satisfiability-checking procedures for propositional classical logic are extended to Kripke structures, and translation-based methods, in which satisfiability of a propositional modal formula is encoded as a first-order formula of a special form, and specialized decision procedures for first-order formulas are used to check the satisfiability of this first-order formula.
2. An alternative characterization of satisfiability by using calculi of a special form, called tableau calculi, gives rise to tableau-based procedures, in which the satisfiability of a formula is checked in a top-down way, by inspecting the inference rules that can be used to derive this formula.

There exist other characterizations of satisfiability for modal logics, for example the matrix characterizations developed in [Wallen 1990], but they have not so far lead to efficient implementations.

Our method, called the *inverse method* following Maslov [1983], uses the characterization of satisfiability in terms of the so-called sequent calculi, developed for classical and nonclassical logics by Gentzen [1934], [see also Goble 1974]. Tableau calculi can often be reformulated as sequent calculi.

In this section we introduce two sequent calculi for K and show how they can be used for proof-search by the tableau method, in the next section we will introduce a similar calculus more suitable for the inverse method.

A *literal* is a propositional variable or its negation. We only consider *modal formulas in negation normal form*, i.e. formulas built from literals using $\wedge$, $\vee$, $\Box$ and $\Diamond$. A formula is called *satisfiable* if it is true in at least one Kripke model. Satisfiability in K can be characterized using a *sequent calculus* introduced below.

We denote propositional variables by lower-case letters, and formulas by upper-case letters. We will use the formula

$$E = \Box a \wedge \Box(\neg a \vee b) \wedge \Diamond\neg b \wedge \Diamond b \qquad (1)$$

for nearly all examples of this paper. We consider conjunction and disjunction right-associative. For example, formula (1) denotes $\Box a \wedge (\Box(\neg a \vee b) \wedge (\Diamond\neg b \wedge \Diamond b))$.

A *sequent* is a finite multiset of formulas. We denote sequents by Greek letters Γ and Δ. We denote the multiset union of two multisets Γ and Δ by Γ, Δ and the addition of an element A to a multiset Γ by Γ, A. For a sequent $\Gamma = A_1, \ldots, A_n$, we denote by $\Diamond\Gamma$ the sequent $\Diamond A_1, \ldots, \Diamond A_n$. A sequent $A_1, \ldots, A_n$ is called *satisfiable* if there exists a Kripke model in which all formulas $A_1, \ldots, A_n$ are true.

It is well-known that satisfiability in K can be characterized using a sequent calculus, i.e. a calculus of a special form that derives sequents.

The *sequent calculus* Seq is given in Figure 1. We call a

- *inference* in Seq any instance of an inference rule;
- *derivation of a sequent* Γ any tree formed by inferences and having Γ as the root;
- *refutation of a sequent* Γ any derivation of Γ whose leaves are axioms.

We will use the notions of inference, derivation and refutation for all other calculi introduced in this paper.

An example refutation in this calculus is given in Figure 2. The connection between derivability in Seq and satisfiability of sequents is give by the following Completeness Theorem.

THEOREM 2.1 (Completeness of Seq) *A sequent S is unsatisfiable if and only if it has a refutation in* Seq.

The proof of this theorem may be found in standard textbooks [e.g. Fitting 1983] or in [Goble 1974].

Tableau-based procedures start with the goal formula, and apply the inference rules of Seq in the direction from the conclusion to the premises. For example, the tableau method can obtain the derivation of Figure 2 using the following sequence of steps:

axioms: $\Gamma, p, \neg p$ $\qquad \dfrac{\Gamma, A, B}{\Gamma, A \wedge B}\,(\wedge)$

$\dfrac{\Gamma, A \quad \Gamma, B}{\Gamma, A \vee B}\,(\vee) \qquad \dfrac{\Delta, A}{\Gamma, \Box\Delta, \Diamond A}\,(\Diamond)$

Figure 1: Sequent calculus Seq

$$\dfrac{\dfrac{\dfrac{a, \neg a, \neg b \quad a, b, \neg b}{a, \neg a \vee b, \neg b}\,(\vee)}{\Box a, \Box(\neg a \vee b), \Diamond\neg b, \Diamond b}\,(\Diamond)}{\Box a \wedge \Box(\neg a \vee b) \wedge \Diamond\neg b \wedge \Diamond b}\,(\wedge), (\wedge), (\wedge)$$

A sequence of inferences $(\wedge)$ is shown as one inference.

Figure 2: Derivation of Formula (1) in Seq

$$\Box a \wedge \Box(\neg a \vee b) \wedge \Diamond\neg b \wedge \Diamond b \qquad \Rightarrow$$

$$\dfrac{\Box a, \Box(\neg a \vee b), \Diamond\neg b, \Diamond b}{\Box a \wedge \Box(\neg a \vee b) \wedge \Diamond\neg b \wedge \Diamond b}\,(\wedge), (\wedge), (\wedge) \qquad \Rightarrow$$

$$\dfrac{\dfrac{a, \neg a \vee b, \neg b}{\Box a, \Box(\neg a \vee b), \Diamond\neg b, \Diamond b}\,(\Diamond)}{\Box a \wedge \Box(\neg a \vee b) \wedge \Diamond\neg b \wedge \Diamond b}\,(\wedge), (\wedge), (\wedge) \qquad \Rightarrow$$

$$\dfrac{\dfrac{\dfrac{a, \neg a, \neg b \quad a, b, \neg b}{a, \neg a \vee b, \neg b}\,(\vee)}{\Box a, \Box(\neg a \vee b), \Diamond\neg b, \Diamond b}\,(\Diamond)}{\Box a \wedge \Box(\neg a \vee b) \wedge \Diamond\neg b \wedge \Diamond b}\,(\wedge), (\wedge), (\wedge)$$

Tableau-based procedures are easy to design. To make them more efficient, a number of optimizations of two kinds are used. One kind is logical optimizations that exploit proof-theoretic properties of Seq, for example invertibility of inference rules, another kind are optimizations of the backtrack search algorithms used by such procedures.

The inverse method discussed in this paper checks goal formulas for satisfiability using proof-search in sequent calculi in the inverse direction, i.e., from the premises of the rules to their conclusions (hence the name "inverse"). First, it builds axioms of the sequent calculus, and then derives more and more complex sequents until a refutation is obtained. For the example above, the inverse method would find a derivation using the following sequence of steps:

$$a, \neg a, \neg b \quad \Rightarrow \quad a, \neg a, \neg b \quad a, b, \neg b \quad \Rightarrow$$

$$\dfrac{a, \neg a, \neg b \quad a, b, \neg b}{a, \neg a \vee b, \neg b}\,(\vee) \qquad \Rightarrow$$

$$\dfrac{\dfrac{a, \neg a, \neg b \quad a, b, \neg b}{a, \neg a \vee b, \neg b}\,(\vee)}{\Box a, \Box(\neg a \vee b), \Diamond\neg b, \Diamond b}\,(\Diamond) \qquad \Rightarrow$$

$$\dfrac{\dfrac{\dfrac{a, \neg a, \neg b \quad a, b, \neg b}{a, \neg a \vee b, \neg b}\,(\vee)}{\Box a, \Box(\neg a \vee b), \Diamond\neg b, \Diamond b}\,(\Diamond)}{\Box a \wedge \Box(\neg a \vee b) \wedge \Diamond\neg b \wedge \Diamond b}\,(\wedge), (\wedge), (\wedge)$$

The idea of the inverse method may seem strange, for example there is an infinite number of possibilities for choosing the axioms, but it can be made feasible by exploiting some proof-theoretic properties of the sequent calculus, most notably the *subformula property*:

LEMMA 2.2 *Every formula occurring in a derivation of a sequent* Γ *consists of subformulas of formulas of* Γ.

The subformula property obviously follows from the completeness theorem for Seq.

The first immediate consequence of the subformula property is that amongst the axioms $\Gamma, p, \neg p$ of Seq only some of them can be used to prove a given goal formula G, namely those in which the literals p, $\neg p$ and every formula in Γ are subformulas of G. Then, we can choose the single representative $p, \neg p$ among all sequents of the form $\Gamma, p, \neg p$. This idea requires a reformulation of Seq in a suitable way explained in the next section.

3 The inverse calculus $\mathcal{H}$

In this section we introduce a sequent calculus suited for a bottom-up proof-search. Such a calculus can be obtained in a straightforward way by modifying Seq.

In the inverse method, the sequent calculus depends on the goal formula G. Such a calculus, denoted by $\mathcal{H}_G$ and an example derivation in this calculus are shown in Figures 3 and 4. Compare this derivation with that of Figure 2.

The completeness of the inverse method is formulated as follows:

$$
\text{axioms: } p, \neg p \qquad \frac{\Gamma, A, A}{\Gamma, A}\,(C)
$$

$$
\frac{\Gamma, A}{\Gamma, A \wedge B}\,(\wedge_l) \qquad \frac{\Gamma, B}{\Gamma, A \wedge B}\,(\wedge_r)
$$

$$
\frac{\Gamma, A \quad \Delta, B}{\Gamma, \Delta, A \vee B}\,(\vee)
$$

$$
\frac{\Gamma, A}{\Box\Gamma, \Diamond A}\,(\Diamond) \qquad \frac{\Gamma}{\Box\Gamma, \Diamond A}\,(\Diamond^+)
$$

In the rules, all formulas occurring in the rules are subformulas of G.

Figure 3: Sequent calculus ꓘ_G

$$
\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{a, \neg a \quad b, \neg b}{a, \neg a \vee b, \neg b}\,(\vee)}{\Box a, \Box(\neg a \vee b), \Diamond\neg b}\,(\Diamond)}{\Box a,\ \ \Box(\neg a \vee b) \wedge \Diamond\neg b \wedge \Diamond b,\ \ \Box(\neg a \vee b) \wedge \Diamond\neg b \wedge \Diamond b}\,(\wedge),(\wedge),(\wedge)}{\Box a, \Box(\neg a \vee b) \wedge \Diamond\neg b \wedge \Diamond b}\,(C)}{\Box a \wedge \Box(\neg a \vee b) \wedge \Diamond\neg b \wedge \Diamond b}\,(\wedge),(\wedge),(C)
$$

Some sequences of inferences are shown as one inference.

Figure 4: Derivation in ꓘ_E

THEOREM 3.1 (Completeness of ꓘ_G) *A formula G is unsatisfiable if and only if it has a refutation in ꓘ_G.*

The proof of this statement is based the following lemma, called the *subsequent lemma*.

LEMMA 3.2 (Subsequent Lemma) *Let a sequent Γ have a refutation in* Seq *and consists of subformulas of a formula G. Then there exists a subsequent $\Delta \subseteq \Gamma$ that has a refutation in ꓘ_G.*

The subsequent lemma can be proved by a straightforward induction on the size of a formula. In this paper we will avoid writing proofs. Instead, we will try to explain ideas of our method in an elementary way. Proofs can be found in [Voronkov 2000].

When we discuss redundancies in ꓘ_G below, we will assert several properties of sequents and derivations in Seq, but not in ꓘ_G. Th Subsequent Lemma allows us to transform such properties of Seq into redundancies of ꓘ_G.

Note that the calculus ꓘ_G has the following properties helpful to organize bottom-up proof-search procedures.

1. Every formula occurring in a derivation is a subformula of G.
2. There exists a finite number of axioms.
3. For every sequent Γ, there exists only a finite number of ways of applying $(\wedge_l)$, $(\wedge_r)$, $(\Diamond)$, $(\Diamond^+)$, or (C) to Γ.
4. For every sequents Γ and Δ, there exists only a finite number of ways of applying $(\vee)$ to Γ and Δ.

A typical proof procedure based on the inverse method uses these properties and works roughly as follows.

1. Let S be a set of sequents, initially $\{p, \neg p \mid p, \neg p \text{ are subformulas of } G\}$.
2. Repeatedly apply all possible inferences to sequents in S, adding their conclusions to S until no new sequents can be obtained or a refutation of G is found.

However, such naive procedures are hopelessly inefficient, unless they are augmented with powerful *redundancy criteria*. There are two kinds of redundancy criteria, allowing us to get rid of *redundant sequents* and *redundant inferences*. Intuitively, a sequent is redundant if it can be removed from the search space without loosing completeness. An inference is redundant if the calculus obtained by dropping this inference is still complete. Very powerful techniques for proving redundancies have been developed in resolution-based calculi for classical logic, [see e.g. Bachmair and Ganzinger 1999, Niewenhuis and Rubio 1999]. The use of such powerful criteria is one of the reasons for the efficiency of the system **TA** [Hustadt and Schmidt 1997, Schmidt 1998] that translates formula into a fragment of first-order logic and uses a first-order theorem prover SPASS that implements many redundancy criteria.

Redundancy criteria can be incompatible. For example, it is possible that some sequents Γ and Δ are redundant, but every refutation contains at least one of these sequents. Therefore, one usually proves completeness for a collection of redundancy criteria. If we prove several redundancy criteria for a calculus, the proof-search by the inverse method works as follows.

1. Let S be a set of sequents, initially

$\{p, \neg p \mid p, \neg p$ are subformulas of $G\}$ *less the redundant sequents of this form.*

2. Repeatedly apply all *nonredundant* inferences to sequents in S, adding to S those conclusions of these inferences *that are nonredundant.*

3. Remove all sequents that *become redundant* due to the addition of these conclusions of inferences.

A number of redundancy criteria for various nonclassical logics are proved in [Voronkov 1992] using a rather general technique. A different (but also very general) technique for proving redundancy criteria is presented in [Voronkov 2000]. Our implementation is based on the latter paper. Some of the redundancies discussed in [Voronkov 2000] will be explained here. Most (but not all) of the redundancies used in our system KЯ can be proved complete by a rather general technique based on proving properties of refutations in Seq and transforming them into redundancy criteria for the inverse method using the Subsequent Lemma.

4 Redundancies

Several redundancy criteria discussed in this paper are based on a particular position of a subformula in the goal formula G. It will be convenient for us to identify subformulas of G with their positions. Therefore, we regard two different occurrences of the same formula in G as different subformulas. In [Voronkov 2000] this is achieved by introducing a special *path calculus* dealing with positions of subformulas in G.

4.1 Modal depth of a subformula

Let us call the *modal depth* of a subformula A of G the number of modalities $\Diamond$ and $\Box$ in G that have A in their scope. For example, if G has the form $\Diamond(\ldots \wedge \Box A)$, then the modal depth of this occurrence of A is two. It is easy to see that there is only one inference rule in Seq that changes the modal depth of subformulas when we go from the conclusion of an inference to its premises, namely $(\Diamond)$, and all formulas in the premise of this rule have the modal depth one greater than those in the conclusion.

LEMMA 4.1 *Let Γ be a sequent occurring in a derivation of G in* Seq. *Then all formulas in Γ have the same modal depth.*

This property immediately gives us the following redundancy criterion: any sequent containing two subformulas of different modal depths is redundant.

4.2 Conflicting subformulas

$\vee$-conflict. Consider the $(\vee)$-rule of Seq:

$$\frac{\Gamma, A \quad \Gamma, B}{\Gamma, A \vee B}\ (\vee).$$

One can prove that the sequents above this inference can either contain A, or B but never both (recall that we identify subformulas with their occurrences in G and treat two different occurrences as different subformulas). Moreover, the same holds for the subformulas of A and B: no sequent in the derivation can contain both a subformula of A and a subformula of B. Therefore, for every subformula $A \vee B$ of Γ we say that each subformula of A *is in $\vee$-conflict* with each subformula of B. Then we claim that any sequent that contains a $\vee$-conflicting pair of subformulas is redundant.

$\Diamond$-conflict. A similar observation can be made for some occurrences of subformulas starting with $\Diamond$. Assume that $\Diamond A$ and $\Diamond B$ are two different subformulas of G of the same modal depth. The only rule of Seq that can introduce a modality $\Diamond$ or $\Box$ into a derivation is the rule $(\Diamond)$:

$$\frac{\Delta, C}{\Gamma, \Box\Delta, \Diamond C}\ (\Diamond).$$

But the conclusion of this rule can only contain *one* subformula $\Diamond A$ or $\Diamond B$ but never both. Therefore, we say that $\Diamond A$ and $\Diamond B$, as well as their subformulas, are in *$\Diamond$-conflict* and again treat any sequent containing a conflicting pair of subformulas as redundant.

Of course, these informal considerations do not *prove* that the combination of redundancy criteria considered so far preserves completeness. This is proved formally in every detail in [Voronkov 2000]. Our aim here is to give the reader an informal understanding of the redundancy criteria used in our system. However, the space limitations does not allow us to discuss the proof or rather sophisticated techniques involved in such completeness proofs.

4.3 Subsumption

We say that a sequent Γ *subsumes* a sequent Δ if $\Gamma \subseteq \Delta$[1]. *Subsumption* is the redundancy criterion that allows one to remove from the search space sequents

[1]This notion of subsumption introduced in resolution-based theorem proving by Robinson [1965] should not be mixed with the notion of subsumption for description logics that has appeared later.

subsumed by other sequents. Therefore, the subsumed sequents are regarded as redundant.

This notion of subsumption is implemented in version 1.0 of our system Kꓘ. In fact, we developed an even stronger notion of subsumption in [Voronkov 1992], which is also compatible with all redundancies discussed in this paper, but we could not yet find a way to efficiently implement the stronger notion.

We will informally explain this stronger notion of subsumption. Let A, B be two subformulas of the goal formula (recall that we identify subformulas with their occurrences). We write $A \geq B$ if this occurrence of B is inside of the occurrence of A (this implies that B is a subformula of A). We say that a sequent Γ $\geq$-subsumes a sequent Δ if and only if for every $A \in \Gamma$ there exists $B \in \Delta$ such that $A \geq B$. It is not hard to argue that, if Γ subsumes Δ, then Γ also $\geq$-subsumes Δ. (Indeed, Γ subsumes Δ if and only if for every $A \in \Gamma$ there exists $B \in \Delta$ such that $A = B$, but $A = B$ implies $A \geq B$).

To illustrate the strength of this notion of subsumption let us make an observation. Consider any inference by $(\wedge_l)$:

$$\frac{\Gamma, A}{\Gamma, A \wedge B}\ (\wedge_l).$$

The conclusion of this inference $\geq$-subsumes the premise, since $A \wedge B \geq A$. Therefore, as soon as this inference is performed, the sequent Γ, A can be removed from the search space. A similar observation can be made for any $(\wedge_r)$-inference. Therefore, for any subformula $A \wedge B$ of the goal, its immediate subformulas A and B are "short-living" and any sequent Δ containing any of these subformulas can be replaced by a sequent $\geq$-subsuming Δ. In Kꓘ this observation is implemented in the following way: the calculus $ꓘ_G$ is modified so it does not contain $(\wedge_l)$ or $(\wedge_r)$-rules at all. Such an optimization in sequent calculi was originally introduced for classical logic in [Voronkov 1985], see also more accessible [. Voronkov 1990]

To prove completeness of $ꓘ_G$ with subsumption a new technique based on so-called *traces* of sequents is developed and explained in [Voronkov 2000].

4.4 Orderings

All redundancy criteria discussed so far introduced redundancy criteria for sequents. Our system Kꓘ also uses a redundancy criterion for inferences based on orderings of subformulas. Consider any nonmodal subformulas of G, for example $A \wedge B$ and $C \vee D$. Suppose that we have obtained a sequent Γ, A, C containing both A and C. There are at least two kinds of inferences of $ꓘ_G$ applicable to Γ, A, C:

$$\frac{\Gamma, A, C}{\Gamma, A \wedge B, C}\ (\wedge) \quad \text{and} \quad \frac{\Gamma, A, C \quad \Delta, D}{\Gamma, \Delta, A, C \vee D}\ (\vee).$$

Our ordering technique allows one to introduce an ordering $\succ$ on subformulas of G and to only allow the rules be applied to minimal subformulas in a sequent. Therefore, if we have $A \succ C$, then the first of the two inference becomes redundant, since the inference is applied to a nonminimal subformula A. Likewise, if we have $C \succ A$, then the second inference becomes redundant.

The completeness of all redundancy criteria considered before this subsection can be proved in a rather straightforward way using the Subsequent Lemma. However, the completeness of the ordering strategy requires special care and rather involved techniques. For example, not every ordering preserves completeness, but there are orderings strong enough so that exactly one subformula will be selected for inferences in any sequent.

5 Experiments

The calculus $ꓘ_G$ for K with the redundancy criteria mentioned above is implemented in version 1.0 of our system Kꓘ (read "K-inverse K"). A preliminary much slower version of KꓘK was implemented in ML and described in [Voronkov 1999]. The current version 1.0 of Kꓘ used for benchmarking is implemented in C++ and comprises about 13,000 lines of C++ code, of which 5,000 lines implement a parser generated automatically using the ANTLR parser generator and PCCTS tool.

In this section we consider the results of running Kꓘ on the collection of tests used for the Tableau'98 competition. We give a comparison with other systems for K. The results for other systems are taken from [Giunchiglia, Giunchiglia and Tacchella 1999, Horrocks, Patel-Schneider and Sebastiani 2000]. The results for other systems were obtained on a computer with the Pentium 350MHz processor using the time limit of 100 seconds. We ran our experiments on a Pentium 450MHz processor and set the limit to 77 seconds, in order to compensate the difference in performance. Some of the results for other systems are from the Tableaux'98 comparison, but some are more recent [see Horrocks et al. 2000].

The experiments were performed on 9 collections of tests. Each collection consists of 21 provable and 21 unprovable formulas of presumably growing difficulty. The results are given in Table 1. p in every column means provable formulas, while n means unprovable. The table contains the number of problems solved by systems within the time limit of 100 seconds. If all 21 problems can be solved, we put > in the table. Table 2 shows the time spent by some systems for solving the most difficult problems in each category.

One can see that Kꓘ 1.0 is weaker than the two best systems ∗**SAT** and **DLP**. However, we would like to note that our current implementation is completely nonoptimized as to proof-search strategies and non-logical optimizations. For example, both ∗**SAT** and **DLP** have strategies for intelligent backtracking and **DLP** implements tabulation. In addition, all top systems have powerful preprocessors that simplify the input formula, while the current version of Kꓘ has no preprocessor. Kꓘ is stronger on this collection of benchmarks than the less optimized implementations **FaCT** of the tableau method, **KSAT** of the SAT-based method, and the earlier versions of **TA** of the first-order translation-based methods. This shows the high potential of future, more optimized versions of Kꓘ.

Our system is based on a systematic proof procedure without any ad hoc optimizations. We expect a considerable speedup by implementing such optimizations. Our expectations are confirmed by the experience of other provers, whose first versions were very considerably slower than the current versions. Among such future optimizations we consider the following.

1. Implementing a formula preprocessor that makes general simplifications of the goal formula and also some specific simplifications that make the inverse proof search easier.

2. Implementing a greedy search for redundancies, similar to the SPASS theorem prover for first-order classical logic [Weidenbach, Afshordel, Brahm, Cohrs, Engel, Keen, Theobalt and Topic 1999] used as a subsystem in **TA**. This means that we give priority to those inferences that can make some sequents from the search space redundant. Currently, Kꓘ behaves like a very naive Davis-Putnam procedure for propositional classical logic.

3. Optimizing the system for the cases when propositional reasoning dominates the modal reasoning (cf. the weak performance of Kꓘ on the *ph* benchmarks in which essentially no modal reasoning is involved).

6 Related works

The inverse method for nonclassical logics. Gentzen [1934] was the first to formulate the inverse method: he has shown how to prove the decidability of intuitionistic propositional logic using the bottom-up search in a sequent calculus based on the subformula property. The remark of Gentzen [1934] remained unnoticed for quite some time. The general interest to the inverse method appeared due to a series of Maslov's works, e.g. [Maslov 1971].

Voronkov [1992] sketches a general scheme of the inverse method for nonclassical logics. Some redundancy criteria considered in this paper are similar to strategies introduced in [Voronkov 1992].

Mints [1993] and Tammet [1994] consider the inverse method for linear logic. Mints [1994] and Tammet [1996] discuss the inverse method for intuitionistic logic, however Tammet's [1996] presentation contains several inaccuracies that are difficult to validate in the absence of proofs. For example, his general scheme of constructing inverse method calculi may result in incomplete calculi, and the intuitionistic calculus of Tammet [1996] is indeed incomplete. The ordering strategy of our paper is similar to a strategy based on the permutability of rules in [Tammet 1996].

Implementation of the inverse method. Tammet [1996] reported some results on the implementation of the inverse method for intuitionistic predicate logic, however it is difficult to comment on his results since calculi underlining his implementation [Tammet 1996] are incomplete.

Our implementation seems to be the first to show that calculi based on forward reasoning may compete with calculi based on backward reasoning for *propositional* logics.

Implementations of other methods. Implementations of other state-of-the-art systems are described in the following papers. SAT-based decision procedures are considered in [Giunchiglia et al. 1998, Tacchella 1999, Giunchiglia et al. 1999]. Hustadt and Schmidt [1997] describe an implementation based on a translation to classical predicate logic and the use of the classical logic theorem prover SPASS [Weidenbach et al. 1999]. Two best tableau-based systems are discussed in [Horrocks and Patel-Schneider 1998, Horrocks 1998, Patel-Schneider 1998]. A comparison of several approaches and a general discussion of evaluation of decision procedures for nonclassical logics can be found in [Hustadt and Schmidt 1997, Hor-

	branch		*d4*		*dum*		*grz*		*lin*		*path*		*ph*		*poly*		*t4p*	
K	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*
leanK 2.0	1	0	1	1	0	0	0	>	>	4	2	0	3	1	2	0	0	0
Crack 1.0	2	1	2	3	3	>	1	>	5	2	2	6	2	3	>	>	1	1
□**KE**	13	3	13	3	4	4	3	1	>	2	17	5	4	3	17	0	0	3
LWB 1.0	6	7	8	6	13	19	7	13	11	8	12	10	4	8	8	11	8	7
Kris	3	3	8	6	15	>	13	>	6	9	3	11	4	5	11	>	7	5
KSAT	8	8	8	5	11	>	17	>	>	3	4	8	5	5	13	12	10	18
FaCT 1.2	6	4	>	8	>	>	>	>	>	>	7	6	6	7	>	>	>	>
Kꓘ	6	7	>	16	>	>	>	>	>	>	15	14	2	2	>	>	>	>
TA	9	9	>	18	>	>	>	>	>	>	20	20	6	9	16	17	>	19
DLP 3.1	19	13	>	>	>	>	>	>	>	>	>	>	7	9	>	>	>	>
∗**SAT** 1.2	>	12	>	>	>	>	>	>	>	>	>	>	8	12	>	>	>	>

Table 1: Results for K

	∗**SAT** 1.2		**DLP** 3.2		**TA** 1.4		Kꓘ	
Test	Size	Time	Size	Time	Size	Time	Size	Time
branch_p	>	0.21	19	46.06	6	65.76	6	29.08
branch_n	12	94.49	13	53.63	6	96.44	7	65.33
d4_p	>	0.06	>	0.05	15	71.11	>	0.09
d4_n	>	2.87	>	1.12	14	44.06	16	69. 17
dum_p	>	0.04	>	0.02	17	64.99	>	0.01
dum_n	>	0.12	>	0.02	16	65.82	>	0
grz_p	>	0.04	>	0.04	>	0.51	>	0.11
grz_n	>	0.01	>	0.05	>	0.33	>	0.08
lin_p	>	0.01	>	0.03	>	9.24	>	0.03
lin_n	>	47.80	>	0.13	>	80.01	>	0.15
path_p	>	0.72	>	0.32	5	25.03	15	36.43
path_n	>	0.96	>	0.36	4	60.84	14	35.92
ph_p	8	48.54	7	10.23	6	43.16	2	0.08
ph_n	12	0.60	>	2.69	9	55.13	2	0.02
poly_p	>	1.73	>	0.11	5	53.48	>	0.54
poly_n	>	2.25	>	0.18	4	9.09	>	0.49
t4p_p	>	0.29	>	0.06	16	88.66	>	0.01
t4p_n	>	1.28	>	0.13	9	87.72	>	0.04

Table 2: Timing Results from [Giunchiglia et al. 1999] and [Horrocks et al. 2000] for ∗**SAT** (options `-k1 -e -m6`), **DLP**, **TA** and Kꓘ.

rocks et al. 2000].

7 Future research

The framework described in this paper is applicable to a number of nonclassical logics, at least K, T, D, $K4$, $T4$, $S4$, their multimodal variants and their variants with global assumptions.

It would be interesting to *extend it to more expressive description logics* [Calvanese, De Giacomo, Lenzerini and Nardi 2000]. An obstacle to such an extension is the way these logics are usually introduced. The technique for designing calculi for various logics developed in the description logic community is essentially oriented to tableau-based procedures. These calculi are formulated in a top-down way and often have explicit annotation for worlds. This simplifies completeness proofs but the resulting calculi do not have obvious proof-theoretical properties to be exploited. As a result, only two kinds of satisfiability checking procedures can be developed: tableau-based ones and semantics-based ones.

To design an inverse-method based calculus one needs to *obtain reformulations of these tableau-oriented calculi in a more traditional way*, where the inference rules are formulated solely in terms of formulas, or otherwise find new techniques for designing bottom-up calculi. Such a logical reformulation is an interesting line of research by itself and can lead to better versions of tableau-based procedures that also exploit proof-theoretical properties of calculi, while the current tableau-based systems are mostly based on various nonlogical optimizations, such as tabulation or intelligent backtracking.

It seems that bottom-up calculi can offer considerable advantages for handling some inference rules. For examples, some description logics have a restricted form of the cut rule:

$$\frac{\Gamma, A \quad \Gamma, \overline{A}}{\Gamma}\ (cut),$$

where A and $\overline{A}$ (equivalent to the negation of A) can be arbitrary subformulas of the goal formula. The problem of using this rule in the direction from the conclusion to the premises is high nondeterminism in selecting the cut formula A. The application in the inverse direction does not meet this problem because Γ, A and $\Gamma, \overline{A}$ must be already derived sequents. Therefore, from the viewpoint of bottom-up calculi there is no essential difference between this rule and, say, the $(\vee)$-rule:

$$\frac{\Gamma, A \quad \Delta, B}{\Gamma, \Delta A \vee B}\ (\vee),$$

in which Γ, A and Δ, B should be already proved sequents.

Another interesting line of research is *combination of top-down and bottom-up proof search*, or *bidirectional search*. In such a combination the inverse method would produce sequents that can be used by the tableau method as additional axioms. This is similar to tabulation, but with an essential difference. The traditional tabulation is used to store *derivable* sequents and checking whether a newly derived sequent is unsatisfiable by verifying whether it is subsumed by a tabulated sequent. If it is, then the newly derived sequent is unsatisfiable, if not, then we have to continue the search for refutation. The inverse method tabulates *all* sequents up to a certain modal depth k, which means that a newly derived sequent of a modal depth $\geq k$ is satisfiable *if and only if* it is not subsumed by any of the tabulated sequent. Therefore, if the subsumption-check fails one should *not* continue search for the proof of this newly derived clause.

A combination of the tableau and the inverse method can also solve the memory problem for the inverse method. On some benchmarks, our implementation of the inverse method uses more than 400M of memory. If one discovers that the memory consumption is too high, one can terminate the inverse proof search and continue the tableau search with the tabulated sequents collected so far.

However, such bidirectional search meets essential difficulties on the theoretical level. The inverse method without redundancies is not efficient at all. The naive tableau proof-search is inefficient as well. When we run bidirectional proof-search, one has to be sure that the optimizations used in the tableau proof-search are compatible with the redundancies used in the inverse proof search. This is a challenging task, since the completeness proofs of the redundancies for the inverse method requires a nontrivial technique.

We feel that a new technique can be used to give model-theoretical completeness proofs of the inverse path calculus with redundancies built in directly into the proof system, similar to completeness proofs for resolution with redundancies [Bachmair and Ganzinger 1997]. More on this point can be found in [Voronkov 2000].

It is not hard to design a procedure which deletes some derived sequents (but when it is necessary proves them again) and runs in polynomial space. However, we doubt that such a procedure would be any efficient in practice. It is interesting to understand if one can design a bottom-up procedure running in polynomial space and having the following property: a sequent Γ is deleted only when it is redundant, so after deletion Γ should not be derived again.

Conclusion

We introduced and implemented a new method of propositional modal reasoning based on the a special bottom-up version of a sequent calculus. Our experiments show that bottom-up procedures, even implemented in a straightforward way, give a viable alternative to existing methods of automated reasoning for modal and description logics.

We hope that the approach undertaken in this paper gives an interesting pattern for implementing decision procedures for PSPACE-complete problems in general and for more expressive description logics.

Acknowledgments

This paper benefited from numerous discussions with (in the alphabetical order) Enrico Franconi, Enrico Giunchiglia, Fausto Giunchiglia, Ian Horrocks, Fabio Massacci, Ulrike Sattler and Renate Schmidt.

The completeness proofs of various redundancies for the sequent calculus were verified by Anatoli Degtyarev. His enthusiastic remarks about the mathematical content of this research helped me to proceed towards the actual implementation.

The system Kꓘ was implemented in a relatively short period of time due to the help from Alexander Riazanov on various aspects of C++ and remarks of Christoph Weidenbach on debugging.

The author is partially supported by a grant from the Faculty of Science and Engineering, University of Manchester.

References

Baader, F. and Hollunder, B. [1991], A terminological knowledge representation system with complete inference algorithms, *in* H. Boley and M. Richter, eds, 'PDK'91', Vol. 567 of *Lecture Notes in Artificial Intelligence*, pp. 67–86.

Bachmair, L. and Ganzinger, H. [1997], A theory of resolution, Research report MPI-I-97-2-005, Max-Planck Institut für Informatik, Saarbrücken, Germany. To appear in Handbook of Automated Reasoning, edited by A. Robinson and A. Voronkov.

Bachmair, L. and Ganzinger, H. [1999], A theory of resolution, *in* A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Elsevier Science and MIT Press. To appear.

Balsiger, P., Heuerding, A. and Schwendimann, S. [1998], Logics Workbench 1.0, *in* H. de Swart, ed., 'Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX'98', Vol. 1397 of *Lecture Notes in Computer Science*, Springer Verlag, Oisterwijk, The Netherlands, p. 35.

Calvanese, D., De Giacomo, G., Lenzerini, M. and Nardi, D. [2000], Reasoning in expressive description logics, *in* A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Elsevier Science and MIT Press. To appear.

Fitting, M. [1983], *Proof methods for modal and intuitionistic logics*, Vol. 169 of *Synthese Library*, Reidel Publ. Comp.

Gentzen, G. [1934], 'Untersuchungen über das logische Schließen', *Mathematical Zeitschrift* **39**, 176–210, 405–431. Translated as [Gentzen 1969].

Gentzen, G. [1969], Investigations into logical deduction, *in* M. Szabo, ed., 'The Collected Papers of Gerhard Gentzen', North Holland, Amsterdam, pp. 68–131. Originally appeared as [Gentzen 1934].

Giunchiglia, E., Giunchiglia, F., Sebastiani, R. and Tacchella, A. [1998], More evaluation of decision procedures for modal logics, *in* A. Cohn, L. Schubert and S. Shapiro, eds, 'Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98)', Morgan Kaufmann, San Francisco, CA, pp. 626–635.

Giunchiglia, E., Giunchiglia, F. and Tacchella, A. [1999], 'SAT-based decision procedures for classical modal logics', *Journal of Automated Reasoning* . To appear in the Special Issues: Satisfiability at the start of the year 2000 (SAT 2000).

Giunchiglia, F. and Sebastiani, R. [1996], Building decision procedures for modal logics from propositional decision procedures: Case study of modal K, *in* M. McRobbie and J. Slaney, eds, 'CADE-13', Vol. 1104 of *Lecture Notes in Computer Science*, pp. 583–597.

Goble, L. [1974], 'Gentzen systems for modal logic', *Notre Dame J. of Formal Logic* **15**, 455–461.

Horrocks, I. [1998], Using an expressive description logic: FaCT or fiction?, *in* A. Cohn, L. Schubert and S. Shapiro, eds, 'Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98)', Morgan Kaufmann, San Francisco, CA, pp. 636–647.

Horrocks, I. and Patel-Schneider, P. [1998], FaCT and DLP, *in* H. de Swart, ed., 'Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX'98', Vol. 1397 of *Lecture Notes in Computer Science*, Springer Verlag, Oisterwijk, The Netherlands, pp. 27–30.

Horrocks, I., Patel-Schneider, P. and Sebastiani, R. [2000], 'An analysis of empirical testing for modal decision procedures', *Logic Journal of the IGPL* . submitted.

Hustadt, U. and Schmidt, R. [1997], On evaluating decision procedures for modal logic, *in* 'IJCAI-97', Vol. 1, pp. 202–207.

Maslov, S. [1971], Proof-search strategies for methods of the resolution, *in* B. Meltzer and D. Michie, eds, 'Machine Intelligence', Vol. 6, American Elsevier, pp. 77–90.

Maslov, S. [1983], An inverse method for establishing deducibility of nonprenex formulas of the predicate calculus, *in* J.Siekmann and G.Wrightson, eds, 'Automation of Reasoning (Classical papers on Computational Logic)', Vol. 2, Springer Verlag, pp. 48–54.

Mints, G. [1993], 'Resolution calculus for the first order linear logic', *Journal of Logic, Language and Information* **2**, 58–93.

Mints, G. [1994], Resolution strategies for the intuitionistic logic, *in* 'Constraint Programming', NATO ASI Series F, Springer Verlag, pp. 289–311.

Niewenhuis, R. and Rubio, A. [1999], Paramodulation-based theorem proving, *in* A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Elsevier Science and MIT Press. To appear.

Patel-Schneider, P. [1998], DLP system description, *in* E. Franconi, G. Giacomo, R. MacGregor, W. Nutt, C. Welty and F. Sebastiani, eds, 'Collected Papers from the International Description Logic Workshop (DL'98)', Vol. 11 of *CEUR-WS*, pp. 87–89.
URL: *http://SunCITE.Informatik.RWTH-Aachen.DE/Publications/CEUR-WS*

Robinson, J. [1965], 'A machine-oriented logic based on the resolution principle', *Journal of the Association for Computing Machinery* **12**(1), 23–41. Reprinted as [Robinson 1983].

Robinson, J. [1983], A machine oriented logic based on the resolution principle, *in* J. Siekmann and G. Wrightson, eds, 'Automation of Reasoning. Classical Papers on Computational Logic', Vol. 1965, Springer, pp. 397–415. Originally appeared as [Robinson 1965].

Schmidt, R. [1998], Resolution is a decision procedure for many propositional modal logics, *in* M. Kracht, M. de Rijke, H. Wansing and M. Zakharyaschev, eds, 'Advances in Modal Logic, I', Vol. 87 of *Lecture Notes*, CSLI Publications, Stanford, pp. 189–208.

Tacchella, A. [1999], *SAT system description, *in* P. Lambrix, A. Borgida, M. Lenzerini, R. Möller and P. Patel-Schneider, eds, 'Proceedings of the 1999 International Description Logic Workshop (DL'99)', Vol. 22 of *CEUR-WS*, pp. 142–144.
URL: *http://SunCITE.Informatik.RWTH-Aachen.DE/Publications/CEUR-WS*

Tammet, T. [1994], 'Proof strategies in linear logic', *Journal of Automated Reasoning* **12**(3), 273–304.

Tammet, T. [1996], A resolution theorem prover for intuitionistic logic, *in* M. McRobbie and J. Slaney, eds, 'CADE-13', Vol. 1104 of *Lecture Notes in Computer Science*, pp. 2–16.

Voronkov, A. [1985], A proof search method (in Russian), Vol. 107 of *Vychislitelnye Sistemy*, Novosibirsk.

Voronkov, A. [1990], A proof-search method for the first order logic, *in* P. Martin-Löf and G. Mintz, eds, 'COLOG'88', Vol. 417 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 327–340.

Voronkov, A. [1992], Theorem proving in non-standard logics based on the inverse method, *in* D. Kapur, ed., '11th International Conference on Automated Deduction', Vol. 607 of *Lecture Notes in Artificial Intelligence*, pp. 648–662.

Voronkov, A. [1999], KK: a theorem prover for K, *in* H. Ganzinger, ed., 'Automated Deduction—CADE-16. 16th International Conference on Automated Deduction', Lecture Notes in Artificial Intelligence, Trento, Italy, pp. 383–387.

Voronkov, A. [2000], How to optimize proof-search in modal logics: a new way of proving redundancy criteria for sequent calculi, Technical Report UMCS-2000-1-1, Department of Computer Science, University of Manchester.
URL: *http://www.cs.man.ac.uk/cstechrep/*

Wallen, L. [1990], *Automated Deduction in Nonclassical Logics*, The MIT Press.

Weidenbach, C., Afshordel, B., Brahm, U., Cohrs, C., Engel, T., Keen, E., Theobalt, C. and Topic, D. [1999], System description: SPASS version 1.0.0, *in* H. Ganzinger, ed., 'Automated Deduction—CADE-16. 16th International Conference on Automated Deduction', Lecture Notes in Artificial Intelligence, Trento, Italy, pp. 378–382.