Implementing Connection Calculi for First-order Modal Logics

Jens Otten

Institut für Informatik, University of Potsdam
August-Bebel-Str. 89, 14482 Potsdam-Babelsberg, Germany
jeotten@cs.uni-potsdam.de

Abstract

This paper presents an implementation of an automated theorem prover for first-order modal logic that works for the constant, cumulative, and varying domain of the modal logics D, T, S4, and S5. It is based on the connection calculus for classical logic and uses prefixes representing world paths and a prefix unification algorithm to capture the restrictions given by the Kripke semantics of the standard modal logics. This permits a modular and elegant treatment of the considered modal logics and yields an efficient implementation. Details of the calculus, the implementation and performance results on the QMLTP problem library are presented.

1 Introduction

Modal logics add the unary modal operators $\Box$ and $\Diamond$ to classical logic. They are used to represent the modalities "it is necessarily true that" and "it is possibly true that", respectively. The *Kripke semantics* of the standard modal logics is defined by a set of worlds and a binary accessibility relation between these worlds. In each single world the classical semantics applies to the standard connectives $\neg$, $\wedge$, $\vee$, $\Rightarrow$, and the modal operators are interpreted with respect to accessible worlds. The properties of the accessibility relation specify the particular modal logic. In *first-order* modal logics *domains* of objects are associated with each world, and the standard universal and existential quantifiers are added. First-order modal logics have many applications, e.g. in planning, natural language processing, and program verification. Many of these application require tools for automated reasoning in order to automate the proof search. Whereas there exist a few automated theorem proving systems for some propositional modal logics, e.g. modleanTAP [1] and MSPASS [9], the availability of even basic implementations for full first-order modal logics is very limited.

Connection calculi are a well-known basis to automate formal reasoning in classical first-order logic [4, 11]. In connection calculi the proof search is guided by connections, i.e. sets of the form $\{P, \neg P\}$ that contain literals with the same predicate symbol but different polarities. This yields a goal-oriented and, therefore, more efficient proof search. Connections correspond to closed branches in the tableau framework [8] or axioms in the sequent calculus [7]. The *matrix characterisation* for classical logic can be seen as the underlying notion of connection calculi for classical logic. This characterization of logical validity can be extended to modal logic by using *prefixes*, which encode the additional non-permutabilities of rule applications in the modal sequent calculus [16, 17].

In this paper an implementation of a connection calculus for the first-order modal logics D, S4, S5, and T is presented. It is based on the leanCoP system for classical first-order logic extended by an algorithm for prefix unification. This paper is structured as follows. In Section 2 the use of prefixes is motivated and the basic definitions required for the matrix characterization for modal logic are introduced. Section 3 presents the algorithm for prefix unification and the connection calculus used for the proof search. Details about the implementation based on this modal connection calculus are given in Section 4. Section 5 presents performance results of the implementation on the QMLTP problem library, before Section 6 concludes with a short summary and some remarks on future work.

2 Preliminaries

The reader is assumed to be familiar with the syntax and semantics of first-order modal logic, see, e.g., [6]. In the following we consider the standard semantics with rigid term designation, i.e. terms denote the same object in every world, and terms are local, i.e. any ground term denotes an existing object in every world. In this paper the letters P, Q are used to denote predicate symbols, f denotes a function symbol, and x, y, z are used to denote (term) variables. Terms are denoted by s, t and are built from functions, constants, and variables. *Atomic formulae*, denoted by A, are built from predicate symbols and terms. The connectives $\neg$, $\wedge$, $\vee$, $\Rightarrow$ denote negation, conjunction, disjunction and implication, respectively. A *(first-order modal) formula*, denoted by F, G, H, consists of atomic formulae, the connectives, the modal operators $\Box$ and $\Diamond$, and the existential and universal quantifiers $\forall x$ and $\exists x$. A *literal*, denoted by L, has the form A or $\neg A$. The complement $\overline{L}$ of a literal L is A if L is of the form $\neg A$, and $\neg A$ otherwise.

2.1 Using Prefixes

In order to adapt the matrix characterization of classical validity to modal logic, the notion of prefixes and a modal substitution are introduced.

While the standard connectives and quantifiers are interpreted as in classical logic, the semantics of the modal operators $\Box$ and $\Diamond$ is defined with respect to a set of worlds. Then, $\Box F$ or $\Diamond F$ are true in a world w, if F is true in *all* worlds accessible from w or *some* world accessible from w, respectively. The properties of the accessibility relation determine the particular modal logic. In the following the modal logics D, T, S4, and S5 are considered. Their accessibility relation is serial (D)[1], reflexive (T), reflexive and transitive (S4), or an equivalence relation (S5), respectively.

The modal sequent calculus extends the classical sequent calculus by the *modal rules* $\Box$*-left*, $\Box$*-right*, $\Diamond$*-left* and $\Diamond$*-right*, which are used to introduce the modal operators $\Box$ and $\Diamond$ into the antecedent (left side) or the succedent (right side) of the sequent, respectively.

Definition 1 (Modal sequent calculus)**.** *The* sequent calculus *for the modal logics D, T, and S4 consists of the axiom and rules of the classical sequent calculus and the four additional inference rules shown in Figure 1 [17].*[2] *It is* $\Gamma_\Box := \{\Box G \,|\, \Box G \in \Gamma\}$*,* $\Delta_\Diamond := \{\Diamond G \,|\, \Diamond G \in \Delta\}$*,* $\Gamma_{(\Box)} := \{G \,|\, \Box G \in \Gamma\}$*, and* $\Delta_{(\Diamond)} := \{G \,|\, \Diamond G \in \Delta\}$*.*

$$\frac{\Gamma^+, F \vdash \Delta^+}{\Gamma, \Box F \vdash \Delta}\ \Box\textit{-left} \qquad \frac{\Gamma^* \vdash F, \Delta^*}{\Gamma \vdash \Box F, \Delta}\ \Box\textit{-right}$$

$$\frac{\Gamma^+ \vdash F, \Delta^+}{\Gamma \vdash \Diamond F, \Delta}\ \Diamond\textit{-right} \qquad \frac{\Gamma^*, F \vdash \Delta^*}{\Gamma, \Diamond F \vdash \Delta}\ \Diamond\textit{-left}$$

logic	Γ^+	Δ^+	Γ^*	Δ^*
D	$\Gamma_{(\Box)}$	$\Delta_{(\Diamond)}$	$\Gamma_{(\Box)}$	$\Delta_{(\Diamond)}$
T	Γ	Δ	$\Gamma_{(\Box)}$	$\Delta_{(\Diamond)}$
S4	Γ	Δ	$\Gamma_\Box$	$\Delta_\Diamond$

Figure 1: The four additional rules of the modal sequent calculus

For the proof search the modal rules are applied upwards, i.e., in an *analytic* way. In that case formulae in the sets Γ^*, Δ^*, Γ^+ and Δ^+ are deleted from the premise. To avoid deleting formulae that are required for a proof the application order of the modal rules need to be controlled.

[1] A relation $R \subseteq W \times W$ is *serial* iff for all $w_1 \in W$ there is some $w_2 \in W$ with $(w_1, w_2) \in R$.

[2] The modal sequent calculus captures the cumulative domain condition (see Section 2.2). There are no similar cut-free sequent calculi (that have the subformula property) for the logics with constant or varying domain or for the modal logic S5.

Example 1 (Application of modal rules)**.** *Let F_1 be the formula $\Box P \Rightarrow \Box\Box P$. The left side of Figure 2 shows a sequent derivation of F_1 in the modal sequent calculus for S4. When the rule $\Box$-left is applied first (in an analytic way), the following application of the rule $\Box$-right deletes the formula P from the antecedent and the derivation cannot be turned into a proof. In the derivation of F_1 in the middle of Figure 2 the two rules $\Box$-right are applied before rule $\Box$-left, and the resulting derivation is a proof.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash P}{\vdash \Box P}\,\Box\text{-}right}{P \vdash \Box\Box P}\,\Box\text{-}right}{\Box P \vdash \Box\Box P}\,\Box\text{-}left}{\vdash \Box P \Rightarrow \Box\Box P} \qquad \begin{array}{ll} w_2 & \dfrac{P \vdash P}{\Box P \vdash P}\,\Box\text{-}left \\ w_2 & \dfrac{\Box P \vdash P}{\Box P \vdash \Box P}\,\Box\text{-}right \\ w_1 & \dfrac{\Box P \vdash \Box P}{\Box P \vdash \Box\Box P}\,\Box\text{-}right \\ w_0 & \dfrac{\Box P \vdash \Box\Box P}{\vdash \Box P \Rightarrow \Box\Box P} \end{array} \qquad \dfrac{\dfrac{\dfrac{\dfrac{P : a_0A_1 \vdash P : a_0a_1a_2}{\Box P : a_0 \vdash P : a_0a_1a_2}\,\Box\text{-}left}{\Box P : a_0 \vdash \Box P : a_0a_1}\,\Box\text{-}right}{\Box P : a_0 \vdash \Box\Box P : a_0}\,\Box\text{-}right}{\vdash \Box P \Rightarrow \Box\Box P : a_0}$$

Figure 2: Sequent derivations of $\Box P \Rightarrow \Box\Box P$ for the modal logic S4

Semantically, every application of the rule $\Box$-*right* introduces a new world accessible from the previous one. Starting with w_0 the derivation in the middle of Figure 2 is labelled with the worlds in which the sequent is forced. $\Box P$ is forced in w_0, w_1 is accessible from w_0 and w_2 is accessible from w_1. As the accessibility relation of S4 is transitive, w_2 is also accessible from w_0. Hence, the formula $\Box P$ on the left side of the sequent is preserved. For the modal logic D, whose accessibility relation is neither reflexive nor transitive, the applications of the rules $\Box$-*left* (or $\Box$-*right*) and $\Box$-*right* would yield the upper sequent $\vdash P$. The $\Box P$ on the left side of the sequent does not survive more than one world transition as the accessibility relation for the modal logic D is not transitive. In the following the notion of prefixes is used in order to name sequences of accessible worlds.

Definition 2 (Prefix)**.** *A* prefix*, denoted by p or q, is a string (sequence of characters) over an alphabet $\mathcal{V} \cup \Pi$, in which $\mathcal{V}$ is a set of* prefix variables *and Π is a set of* prefix constants*.*

To distinguish between elements of $\mathcal{V}$ and elements of Π within a prefix, elements of $\mathcal{V}$ are written in capitals, denoted by V or V_i, whereas small letters, denoted by a or a_i, are used to represent elements of Π. u, w denote strings, ε denotes the empty string, $u \circ w$ denotes the concatenation of u and w. $u \preceq w$ holds if, and only if, u is an initial substring of w or $u = w$.

Example 2 (Prefixes for worlds)**.** *In the right derivation of F_1 in Figure 2 each formula is labelled with a prefix encoding the world in which the formula is forced. Starting with a_0 each application of a modal rule extends the prefix by one character. In the logic S4, A_1 could denote any world accessible from a_0. To ensure that the propositions P on both sides of the sequent are forced in the same world, their two prefixes a_0A_1 and $a_0a_1a_2$ need to refer to the same world. This is achieved by assigning the string a_1a_2 to A_1. In the logic D, A_1 can only denote a world directly accessible from a_0. Hence, only the string a_1 can be assigned to A_1 and the two prefixes a_0A_1 and $a_0a_1a_2$ cannot denote the same world.*

Semantically, a prefix denotes a specific world in a model [5, 17]. Prefixes of literals that form an axiom in the sequent calculus need to denote the same world. Proof-theoretically, a prefix of a formula F captures the modal context of F and specifies the sequence of modal rules that have to be applied (analytically) in order to obtain F in the sequent. Prefix variables and constants represent applications of the modal rules $\Box$-*left*/$\Diamond$-*right* and $\Box$-*right*/$\Diamond$-*left*, respectively. When a modal rule that corresponds to an element in Π (and $\mathcal{V}$ for the modal logic D) is applied, only formulae whose prefix ends with an element of $\mathcal{V}$ are preserved in the premise of the sequent. Hence, the application of the modal rules represented by elements of $\mathcal{V}$ and Π need to be controlled. In order to preserve two atomic formulae that form an axiom in the sequent calculus their prefixes need to be made identical. This is done by a modal substitution that maps elements of $\mathcal{V}$ to strings over $\mathcal{V} \cup \Pi$.

Definition 3 (Modal substitution)**.** *A* modal substitution *is a mapping $\sigma_M : \mathcal{V} \to (\mathcal{V} \cup \Pi)^*$ that assigns a string over the alphabet $\mathcal{V} \cup \Pi$ to every element in $\mathcal{V}$. For the modal logics D and T the* accessibility condition $|\sigma_M(V)| = 1$ *or* $|\sigma_M(V)| \leq 1$ *has to hold for all $V \in \mathcal{V}$, respectively. $\sigma_M(p)$ denotes the string p in which each prefix variable V is replaced by its image $\sigma_M(V)$.*

Substitutions σ_M are assumed to be idempotent, i.e. $\sigma_M \circ \sigma_M = \sigma_M$ holds. They are represented by the set $\{V \backslash p \mid \sigma_M(V) = p \text{ and } p \neq V\}$. The accessibility condition encodes the characteristics of the different modal rules for each logic. For the modal logic D a formula that corresponds to a prefix variable will only survive one application of the modal rules. Hence, only strings of length one can be assigned to prefix variables. For the modal logic T a formula that corresponds to a prefix variable will only survive one application of the modal rules $\Box$*-right* or $\Diamond$*-left*, but any number of applications of the rules $\Box$*-left* and $\Diamond$*-right*. Therefore, strings of length one or ε can be assigned to prefix variables. For the modal logic S4 a formula that corresponds to a prefix variable will survive any number of applications of modal rules. Hence, there is no restriction on strings that can be assigned to prefix variables.

Example 3 (Modal substitution)**.** *Consider the prefixes $a_0 A_1$ and $a_0 a_1 a_2$ of the two literals occurring in the upper sequent of the right derivation in Figure 2. $\sigma_{M1} = \{A_1 \backslash a_1 a_2\}$ is a modal substitution for the logic S4 (but not for D or T) with $\sigma_{M1}(a_0 A_1) = a_0 a_1 a_2 = \sigma_{M1}(a_0 a_1 a_2)$.*

2.2 Matrix Characterization

The matrix characterization of validity for modal logic can be seen as the basis of the connection calculus presented in Section 3.2. As the core implementation of the connection calculus (introduced in Section 4.2) uses a clausal form, the original (non-clausal) matrix characterization [17] is slightly modified. Matrices use a clausal form and the definitions of prefixes and paths are simplified by using the formal definition of a prefixed matrix and an extended Skolemization technique.

A *polarity* $pol \in \{0, 1\}$ is used to represent negation in a matrix, i.e. literals of the form A and $\neg A$ are represented by A^0 and A^1, respectively. A *prefixed formula* $F^{pol}{:}p$ is a formula F marked with a polarity pol and a prefix p. A *(prefixed) clause*, denoted by C, is a set $\{L_1 : p_1, \ldots, L_n : p_n\}$ with literals L_i and prefixes p_i. $F[x \backslash t]$ denotes the formula F in which all free occurrences of x are replaced by t.

Definition 4 (Prefixed matrix)**.** *The* prefixed matrix $M(F^{pol}{:}p)$ *of a prefixed formula $F^{pol}{:}p$ is a set of prefixed clauses. It is defined inductively according to Table 1. The following abbreviation is used: $M_G \cup_\beta M_H := \{C_G \cup C_H \mid C_G \in M_G, C_H \in M_H\}$. x^* is a new term variable, t^* is the Skolem term $f^*(x_1, \ldots, x_n)$ in which f^* is a new function symbol and $x_1, \ldots, x_n$ are all free term and prefix variables in $\forall x G$ or $\exists x G$. V^* is a new prefix variable, a^* is a prefix constant of the form $f^*(x_1, \ldots, x_n)$ in which f^* is a new function symbol and $x_1, \ldots, x_n$ are all free term and prefix variables in $\Box G$ or $\Diamond G$. The prefix of x^* and t^*, denoted pre(x^*) and pre(t^*), is p. The (prefixed) matrix $M(F)$ of a (first-order modal) formula F is the prefixed matrix $M(F^0 : \varepsilon)$.*

In the *graphical representation* of a matrix, its clauses are arranged horizontally, while the literals of each clause are arranged vertically. In order to simplify the implementation *Skolemization* is not only used for Eigenvariables but also for prefix constants. A similar approach is already used for intuitionistic logic [10]. Thus, the irreflexivity test of the reduction ordering [17] is realized by the occur check of the term and prefix unification procedures. The same Skolem function symbol is used for instances of the same subformula, a technique that is similar to the liberalized δ^+-rule in classical tableau calculi [8].

Example 4 (Prefixed matrix)**.** *Let F_1 be the modal formula $(\Diamond \exists x P x \wedge \Box \forall y (\Diamond P y \Rightarrow Q y)) \Rightarrow \Diamond \exists z Q z$. The prefixed matrix of F_1 is $M_1{:=}M(F_1) = \{\{P^1 d : a_1\}, \{P^0 y : V_1 V_2, Q^1 y : V_1\}, \{Q^0 z : V_3\}\}$. x is an Eigenvariable, y, z are free term variables, $a_1 \in \Pi$ is a prefix constant, $V_1, V_2, V_3 \in \mathcal{V}$ are prefix variables. d, a_1 are Skolem functions/constants. The graphical representation of M_1 is shown in Figure 3.*

Table 1: Matrix $M(F^{pol}:p)$ of a first-order modal formula $F^{pol}:p$

type	$F^{pol}:p$	$M(F^{pol}:p)$
atomic	$A^0:p$	$\{\{A^0:p\}\}$
	$A^1:p$	$\{\{A^1:p\}\}$
α	$(\neg G)^0:p$	$M(G^1:p)$
	$(\neg G)^1:p$	$M(G^0:p)$
	$(G\wedge H)^1:p$	$M(G^1{:}p)\cup M(H^1{:}p)$
	$(G\vee H)^0:p$	$M(G^0{:}p)\cup M(H^0{:}p)$
	$(G\Rightarrow H)^0{:}p$	$M(G^1{:}p)\cup M(H^0{:}p)$
ν	$(\Box G)^1:p$	$M(G^1:p\circ V^*)$
	$(\Diamond G)^0:p$	$M(G^0:p\circ V^*)$

type	$F^{pol}:p$	$M(F^{pol}:p)$
β	$(G\wedge H)^0:p$	$M(G^0{:}p)\cup_\beta M(H^0{:}p)$
	$(G\vee H)^1:p$	$M(G^1{:}p)\cup_\beta M(H^1{:}p)$
	$(G\Rightarrow H)^1{:}p$	$M(G^0{:}p)\cup_\beta M(H^1{:}p)$
γ	$(\forall xG)^1:p$	$M(G[x\backslash x^*]^1:p)$
	$(\exists xG)^0:p$	$M(G[x\backslash x^*]^0:p)$
δ	$(\forall xG)^0:p$	$M(G[x\backslash t^*]^0:p)$
	$(\exists xG)^1:p$	$M(G[x\backslash t^*]^1:p)$
π	$(\Box G)^0:p$	$M(G^0:p\circ a^*)$
	$(\Diamond G)^1:p$	$M(G^1:p\circ a^*)$

Connections and paths are the basic concepts of the matrix characterization. They correspond to axioms and atomic sequents (that cannot be reduced anymore) in the sequent calculus, respectively.

Definition 5 (Connection, path, term substitution)**.** *The set $\{P^1(s_1,\ldots,s_n):p_1, P^0(t_1,\ldots,t_n):p_2\}$ is a* connection*. It consists of a pair of literals with the same predicate symbol but different polarities. A* path *pt through a matrix $M=\{C_1,\ldots,C_n\}$ is a set of literals that contains one literal from each clause, i.e. $pt := \cup_{i=1}^{n}\{L_i\}$ with $L_i\in C_i$. A* term substitution *σ_Q is a mapping from the set of term variables to the set of terms. In $\sigma_Q(L)$ all term variables x in L are substituted by their image $\sigma_Q(x)$.*

Example 5 (Connection, path, term substitution)**.** *Consider the matrix M_1 of Example 4 and its graphical representation in Figure 3. Then, $\{P^1d:a_1, P^0y:V_1V_2\}$ and $\{Q^1y:V_1, Q^0z:V_3\}$ are connections. $\{P^1d:a_1, P^0y:V_1V_2, Q^0z:V_3\}$ and $\{P^1d:a_1, Q^1y:V_1, Q^0z:V_3\}$ are (the only) paths through M_1. σ_{Q_1} with $\sigma_{Q_1}(y)=d$ and $\sigma_{Q_1}(z)=d$ is a term substitution.*

To ensure that a connection represents an axiom in the sequent calculus, the atomic formulae in the connection have to unify under a term substitution. Furthermore, their prefixes have to unify under a prefix substitution. This ensures that these formulae are not deleted by applications of modal rules.

Definition 6 (Admissible substitution, complementarity)**.** *Let $\mathcal{L}\in\{$D, T, S4, S5$\}$ be a modal logic, $\mathcal{D}\in\{$constant, cumulative, varying$\}$ be a domain, σ_M be a modal substitution for $\mathcal{L}$ and σ_Q be a term substitution. The* (combined) substitution *$\sigma := (\sigma_Q,\sigma_M)$ is* admissible *iff the domain condition in Table 2 holds for all term variables x and (I) for cumulative domain for all Skolem terms t in $\sigma_Q(x)$, or (II) for varying domain for all term variables/Skolem terms s in $\sigma_Q(x)$. A connection $\{L_1:p_1, L_2:p_2\}$ is σ-*complementary *for a substitution $\sigma=(\sigma_Q,\sigma_M)$ iff $\sigma_Q(\sigma_M(L_1:p_1))=\sigma_Q(\sigma_M(\overline{L_2}:p_2))$. For the modal logic S5 only the last character of all prefixes, i.e. of pre(x), pre(s), p_1 and p_2, is considered.*

Table 2: Domain condition for the substitution $\sigma=(\sigma_Q,\sigma_M)$

	logic $\mathcal{L}$	
domain $\mathcal{D}$	D, T, S4	S5
constant	–	–
cumulative	$\sigma_M(pre(t))\preceq\sigma_M(pre(x))$	–
varying	$\sigma_M(pre(s))=\sigma_M(pre(x))$	$\sigma_M(pre(s))=\sigma_M(pre(x))$

$$\left[\begin{array}{ccc} \left[\begin{array}{c} P^1 d : a_1 \\ \\ \end{array}\right] & \left[\begin{array}{c} P^0 y : V_1 V_2 \\ Q^1 y : V_1 \end{array}\right] & \left[\begin{array}{c} \\ Q^0 z : V_3 \end{array}\right] \end{array}\right]$$

Figure 3: Graphical representation of the matrix M_1

The constant domain variants place no restriction on the modal substitution as the same objects exist in every world. For the cumulative domain variants any Skolem term t introduced by a quantifier rule in a world w_1 exists in a world w_2 that is directly or indirectly accessible from w_1. Hence, if t occurs in $\sigma_Q(x)$ then the prefix $w_1 = \sigma_M(pre(t))$ has to be an initial string of $w_2 = \sigma_M(pre(x))$.[3] In case of the varying domain variants objects may only exists in the world in which they are introduced. Hence, if s occurs in $\sigma_Q(x)$ then the prefixes $w_1 = \sigma_M(pre(s))$ and $w_2 = \sigma_M(pre(x))$ need to be identical.

Example 6 (Admissible substitution, complementarity)**.** *Consider the matrix M_1 of Example 4 with $\sigma_{Q_1}(y) = \sigma_{Q_1}(z) = d$, $\sigma_{M1}(V_1) = \sigma_{M1}(V_3) = a_1$, and $\sigma_{M1}(V_2) = \varepsilon$ (for T, S4) or $\sigma_{M1}(V_2) = a_1$ (for S5). Then, σ_{M1} is a modal substitution for the modal logics T, S4, and S5, but not for D. The substitution $\sigma_1 = (\sigma_{Q_1}, \sigma_{M1})$ is admissible for the constant, cumulative, and varying domain as $pre(y) = pre(z) = pre(d) = a_1$. The connections $\{P^1 d : a_1, P^0 y : V_1 V_2\}$ and $\{Q^1 y : V_1, Q^0 z : V_3\}$ are σ_1-complementary.*

The notion of *multiplicity* is used to encode the number of clause copies used in a proof. It is a function $\mu : M \to \mathbb{N}$ that assigns a natural number to each clause in M specifying how many copies of this clause are considered in a proof. In the *copy of a clause* C every (term and prefix) variable in C is replaced by a unique new variable. M^μ denotes the matrix that includes these clause copies. Clause copies correspond to applications of the contraction rule in the sequent calculus.

Theorem 1 (Matrix characterisation for modal logic)**.** *Let $\mathcal{L} \in \{D,T,S4,S5\}$ be a modal logic and $\mathcal{D} \in \{constant,cumulative,varying\}$ be a domain. A formula F is valid in the logic $\mathcal{L}$ with domain $\mathcal{D}$ iff there is a multiplicity μ, an admissible substitution $\sigma = (\sigma_Q, \sigma_M)$ for $\mathcal{L}/\mathcal{D}$, and a set of σ-complementary connections such that every path through $M(F)^\mu$ contains a connection from this set.*

Example 7 (Matrix characterisation for modal logic)**.** *Consider the formula F_1 of Example 4 and the graphical representation of its matrix M_1 in Figure 3. Let $\mu \equiv 1$, $\sigma_{Q_1}(y) = \sigma_{Q_1}(z) = d$, $\sigma_{M1}(V_1) = \sigma_{M1}(V_3) = a_1$, and $\sigma_{M1}(V_2) = \varepsilon$ (for T, S4) or $\sigma_{M1}(V_2) = a_1$ (for S5). $\sigma_1 = (\sigma_{Q_1}, \sigma_{M1})$ is admissible for all domains, and every path through M_1^μ contains a σ_1-complementary connection. Hence, F_1 is valid in the modal logics T, S4, and S5 with constant, cumulative, and varying domain.*

3 Proof Search

An effective way to carry out proof search that is based on the matrix characterization is done with a connection-driven search strategy. For this purpose the connection calculus is extended by additional prefixes. Furthermore, a prefix unification algorithm is required in order to calculate the modal substitution by unifying the prefixes of the literals in each connection.

3.1 Prefix Unification

Formally, prefix unification is the problem to find a modal substitution or *unifier* σ_M for a set of prefix equations $E = \{p_1 = q_1, \ldots, p_n = q_n\}$ such that $\sigma_M(p_i) = \sigma_M(q_i)$ for all $1 \leq i \leq n$. A set of unifiers Σ is a set of *most general unifiers* for E if, and only if, every unifier τ is an instance of some $\sigma \in \Sigma$ (completeness) and no unifier $\sigma \in \Sigma$ is an instance of another unifier $\tau \in \Sigma$ (minimality).

[3] This condition — as well as the (non-applicable) accessibility condition for S5 — are slightly corrected conditions of [17].

As prefixes are strings, general algorithms for string unification [4] could be used for unifying prefixes. But for general string unification the number of most general unifiers might not be finite. A more efficient unification takes the *prefix property* of all prefixes $p_1, p_2, \ldots$ into account: for all prefixes $p_i = u_1 X w_1$ and $p_j = u_2 X w_2$ with $X \in \mathcal{V} \cup \Pi$ the property $u_1 = u_2$ holds. The prefix property reflects the fact that the order in which modal rules are applied corresponds to a (formula) tree structure.

Depending on the modal logic the accessibility condition (see Definition 3) has to be respected when calculating the modal substitution σ_M, i.e. for all $V \in \mathcal{V}$: $|\sigma_M(V)| = 1$ for the modal logic D, $|\sigma_M(V)| \leq 1$ for the modal logic T and no restriction for the modal logics S4 and S5. The prefix unification for D is a simple pattern matching, i.e. the standard term unification can be used. For S4 the prefix unification for intuitionistic logic can be used; see [10] for details. For S5 only the last character of each prefix (or ε if the prefix is ε) has to be unified. The prefix unification for T is specified by a set of rewriting rules. Given a prefix equation $E = \{p = q\}$ it calculates a set of most general unifiers for E. For an equation of the form $V_1 u = V_2 w$ the substitution $\{V_1 \backslash \varepsilon, V_2 \backslash \varepsilon\}$ would not be minimal (but an instance of $\{V_1 \backslash V_2\}$). To prevent the calculation of such substitutions, the right side of each equation is split by a bar. Then, V_2 is moved to the left side of the bar, i.e., $u = V_2 | w$, when ε is assigned to V_1.

Definition 7 (Prefix unification)**.** *The* prefix unification *of two prefixes p and q for the modal logic T is done by applying the rewriting rules in Table 4; it is $V \in \mathcal{V}$, $a \in \Pi$, $X \in \mathcal{V} \cup \Pi$, $u, w \in (\mathcal{V} \cup \Pi)^*$. The application of a rewriting rule replaces a tuple (E, σ_M) by a tuple $(E', \sigma_M{}')$. E and E' are prefix equations, σ_M and $\sigma_M{}'$ are modal substitutions. The unification starts with the tuple $(\{p = \varepsilon | q\}, \{\})$ and terminates when the tuple $(\{\}, \sigma_M)$ is derived. In this case σ_M represents a most general unifier. Rules can be applied non-deterministically as there might be more than one most general unifier.*

R1.	$\{\varepsilon = \varepsilon \| \varepsilon\}, \sigma_M$	$\rightarrow$	$\{\}, \sigma_M$	
R2.	$\{\varepsilon = \varepsilon \| X w\}, \sigma_M$	$\rightarrow$	$\{X w = \varepsilon \| \varepsilon\}, \sigma_M$	
R3.	$\{V u = \varepsilon \| \varepsilon\}, \sigma_M$	$\rightarrow$	$\{u = \varepsilon \| \varepsilon\}, \sigma_M \cup \{V \backslash \varepsilon\}$	
R4.	$\{X u = \varepsilon \| X w\}, \sigma_M$	$\rightarrow$	$\{u = \varepsilon \| w\}, \sigma_M$	
R5.	$\{V u = \varepsilon \| X w\}, \sigma_M$	$\rightarrow$	$\{V u = X \| w\}, \sigma_M$	$(X \neq V)$
R6.	$\{a u = \varepsilon \| V w\}, \sigma_M$	$\rightarrow$	$\{V w = a \| u\}, \sigma_M$	
R7.	$\{V_1 u = \varepsilon \| V_2 w\}, \sigma_M$	$\rightarrow$	$\{w = V_1 \| u\}, \sigma_M \cup \{V_2 \backslash \varepsilon\}$	$(V_1 \neq V_2)$
R8.	$\{V u = X \| w\}, \sigma_M$	$\rightarrow$	$\{u = X \| w\}, \sigma_M \cup \{V \backslash \varepsilon\}$	
R9.	$\{V u = X \| w\}, \sigma_M$	$\rightarrow$	$\{u = \varepsilon \| w\}, \sigma_M \cup \{V \backslash X\}$	
R10.	$\{a u = V \| w\}, \sigma_M$	$\rightarrow$	$\{u = \varepsilon \| w\}, \sigma_M \cup \{V \backslash a\}$	

Figure 4: Prefix unification for the modal logic T

To solve a *set* of prefix equations $E = \{p_1 = p_1, \ldots, q_n = t_q\}$ the equations in E are solved one after the other. The modal substitution σ_M calculated for the first equation is applied to E and the next equation in E is considered and so on. Different substitutions σ_M are calculated on backtracking.

Example 8 (Prefix unification)**.** *Consider the connection $\{P^1 d : a_1, P^0 y : V_1 V_2\}$ from Example 5 and 6. The most general unifiers for its two prefixes a_1 and $V_1 V_2$ are calculated as follows:* $\{a_1 = \varepsilon | V_1 V_2\}, \{\} \xrightarrow{R6} \{V_1 V_2 = a_1 | \varepsilon\}, \{\} \xrightarrow{R8} \{V_2 = a_1 | \varepsilon\}, \{V_1 \backslash \varepsilon\} \xrightarrow{R9} \{\varepsilon = \varepsilon | \varepsilon\}, \{V_1 \backslash \varepsilon, V_2 \backslash a_1\} \xrightarrow{R1} \{\}, \{V_1 \backslash \varepsilon, V_2 \backslash a_1\}$ *and* $\{a_1 = \varepsilon | V_1 V_2\}, \{\} \xrightarrow{R6} \{V_1 V_2 = a_1 | \varepsilon\}, \{\} \xrightarrow{R9} \{V_2 = \varepsilon | \varepsilon\}, \{V_1 \backslash a_1\} \xrightarrow{R3} \{\varepsilon = \varepsilon | \varepsilon\}, \{V_1 \backslash a_1, V_2 \backslash \varepsilon\} \xrightarrow{R1} \{\}, \{V_1 \backslash a_1, V_2 \backslash \varepsilon\}$. *Hence, the two most general unifiers are $\{V_1 \backslash \varepsilon, V_2 \backslash a_1\}$ and $\{V_1 \backslash a_1, V_2 \backslash \varepsilon\}$.*

[4] This is also called the *monoid* problem; it is the equation theory in which there is a neutral element ε and the associativity of the string concatenation operator $\circ$ holds.

3.2 The Connection Calculus

The matrix *characterization* of Theorem 1 can serve as the basis for a proof *calculus*. Checking that all paths contain a σ-complementary connection can be done by, e.g., sequent or tableau calculi. The connection calculus uses a *connection-driven* search strategy in order to determine an appropriate set of connections. It is already successfully used for automated theorem proving in classical and intuitionistic logic [11, 12]. Proof search in the connection calculus starts by selecting a start clause. Afterwards, connections are successively identified in order to make sure that all paths through the matrix contain a σ-complementary connection for some combined substitution $\sigma = (\sigma_Q, \sigma_M)$. This process is guided by an *active path*, a subset of a path through the matrix. The multiplicity μ is increased dynamically during the path checking process. The connection calculus is adapted to modal logic by simply adding prefixes to all literals and using the prefix unification algorithm presented in Section 3.1.

Definition 8 (Modal connection calculus)**.** *The axiom and the rules of the* modal connection calculus *are given in Figure 5. The words of the calculus are tuples of the form "$C, M, Path$", where M is a (prefixed) matrix, C and $Path$ are sets of (prefixed) literals or ε. C is called the* subgoal clause *and $Path$ is called the* active path*. C_1 and C_2 are clauses, $\sigma = (\sigma_Q, \sigma_M)$ is an admissible substitution, $\{L_1 : p_1, L_2 : p_2\}$ is a σ-complementary connection. The substitutions σ_Q and σ_M are global (or* rigid*), i.e. they are applied to the whole derivation.*

Axiom (A) $\dfrac{}{\{\}, M, Path}$

Start (S) $\dfrac{C_2, M, \{\}}{\varepsilon,\ M,\ \varepsilon}$ and C_2 is copy of $C_1 \in M$

Reduction (R) $\dfrac{C, M, Path \cup \{L_2 : p_2\}}{C \cup \{L_1 : p_1\}, M, Path \cup \{L_2 : p_2\}}$ and $\{L_1 : p_1, L_2 : p_2\}$ is σ-complementary

Extension (E) $\dfrac{C_2 \setminus \{L_2 : p_2\}, M, Path \cup \{L_1 : p_1\} \qquad C, M, Path}{C \cup \{L_1 : p_1\}, M, Path}$ and C_2 is a copy of $C_1 \in M$, $L_2 : p_2 \in C_2$, $\{L_1 : p_1, L_2 : p_2\}$ is σ-complementary

Figure 5: The connection calculus for modal logic

A derivation for $C, M, Path$ with the admissible substitution $\sigma = (\sigma_Q, \sigma_M)$ for the logic $\mathcal{L}$ and the domain $\mathcal{D}$ in which all leaves are axioms is called a *modal connection proof* for $C, M, Path$. A *modal connection proof* for the matrix M is a modal connection proof for $\varepsilon, M, \varepsilon$.

Theorem 2 (Correctness and completeness)**.** *Let $\mathcal{L} \in \{$D, T, S4, S5$\}$ be a modal logic and $\mathcal{D} \in \{$constant, cumulative, varying$\}$ be a domain. A modal first-order formula F is valid in the modal logic $\mathcal{L}$ with domain $\mathcal{D}$ iff there is a modal connection proof for $M(F)$ for the logic/domain $\mathcal{L}/\mathcal{D}$.*

The proof of Theorem 2 is based on the the matrix characterization for modal logic (see Theorem 1) and the correctness and completeness of the connection calculus [4]. Proof search in the clausal connection calculus is carried out by applying the rules of the calculus in an analytic way, i.e. from bottom to top. During the proof search backtracking might be required, i.e. alternative rules need to be considered if the chosen rule (instance) does not lead to a proof. Alternative applications of rules occur whenever

more than one rule or more than one instance of a rule can be applied, i.e. when choosing the clause C_1 in the start and extension rule or the literal $L_2 : p_2$ in the reduction and extension rule. No backtracking is required when choosing the literal $L_1 : p_1$ in the reduction or extension rule as all literals in C are considered in subsequent proof steps anyway. The term substitution σ_Q and the modal substitution σ_M are calculated by algorithms for term unification and prefix unification (see Section 3.1), respectively, whenever a reduction or extension rule is applied.

Example 9 (Modal connection calculus)**.** *Consider the formula F_1 and its matrix $M_1 = \{\{P^1 d : a_1\}, \{P^0 y : V_1 V_2, Q^1 y : V_1\}, \{Q^0 z : V_3\}\}$ of Example 4. A derivation for M_1 in the modal connection calculus with $\sigma_Q(z') = \sigma_Q(z') = d$, $\sigma_M(V_1') = \sigma_M(V_3') = a_1$ and $\sigma_M(V_2') = \varepsilon$ (for T, S4) or $\sigma_M(V_2') = a_1$ (for S5) is shown in Figure 6. y', z' and V_1', V_2', V_3' are new term and prefix variables. The two extension steps use the connections $\{P^1 d : a_1, P^0 y' : V_1' V_2'\}$ and $\{Q^1 y' : V_1', Q^0 z' : V_3'\}$. As all leaves are axioms and the substitution $\sigma_1 = (\sigma_Q, \sigma_M)$ is admissible the derivation is also a proof for M_1. Hence, the*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\{\}, M_1, \{P^1 d : a_1, Q^1 y' : V_1'\}}\,A \qquad \dfrac{}{\{\}, M_1, \{P^1 d : a_1\}}\,A}{\{Q^1 y' : V_1'\}, \{\{P^1 d : a_1\}, \{P^0 y : V_1 V_2, Q^1 y : V_1\}, \{Q^0 z : V_3\}\}, \{P^1 d : a_1\}}\,E \quad \dfrac{}{\{\}, M_1, \{\}}\,A}{\{P^1 d : a_1\}, \{\{P^1 d : a_1\}, \{P^0 y : V_1 V_2, Q^1 y : V_1\}, \{Q^0 z : V_3\}\}, \{\}}\,E}{\varepsilon, \{\{P^1 d : a_1\}, \{P^0 y : V_1 V_2, Q^1 y : V_1\}, \{Q^0 z : V_3\}\}, \varepsilon}\,S$$

Figure 6: A proof for M_1 in the modal connection calculus

formula F_1 is valid in the modal logics T, S4, and S5 (with constant, cumulative, and varying domain). The proof of F_1 using the graphical matrix representation is depicted in Figure 7. The literals of each connection are connected with a line. The literals of the active path are boxed. In the first step the first clause $\{P^1 d : a_1\}$ is selected as start clause. The second step is an extension step to the first literal of the second clause. The second step is an extension step to the third clause. Variables in clauses that are used in a proof step are renamed.

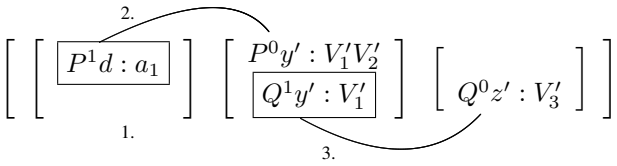

Figure 7: The modal connection proof for M_1 using the graphical matrix representation

Additional techniques to prune the search space in the connection calculus are regularity, lemmata, restricted backtracking, and the use of a definition clausal form translation; see [12] for details. All these techniques can be integrated into the modal connection calculus as well.

4 An Implementation

The connection calculus for modal logic presented in Section 3.2 was implemented in Prolog. The main components are the connection-driven proof search procedure and the prefix unification algorithm. The complete source code can be obtained at `http://www.leancop.de/mleancop/`.

4.1 Prefix Unification

In the Prolog implementation strings are represented by Prolog lists, e.g. the prefix $a_0 a_1 V_2$ is represented by the list `[a0,a1,V2]`. Prefix variables are Prolog variables, i.e. they start with a capital, prefix constants are Prolog atoms, i.e. they start with a small letter.

The source code of the prefix unification for the modal logic T is shown in line *R1* to *R10* of Figure 8. Each Prolog clause corresponds exactly to one of the rewrite rules defined in Figure 4. The predicate `tuni_t(S,[],T)` succeeds if the two prefixes `S` and `T` can be unified with respect to the accessibility condition for the modal logic T given in Definition 3. In this case the prefix variables are instantiated with a most general unifier for `S` and `T`. Alternative unifiers are calculated via backtracking. According to the accessibility condition for the modal logic T, prefix variables may only be instantiated with the empty string or a single prefix constant or variable. As the Skolemization technique is applied to *prefix* constants as well, a term unification with the Prolog predicate `unify_with_occurs_check` is required (line *R4*, *R9* and *R10*) whenever a prefix constant needs to be unified with another prefix constant or variable.

```
 (a)  prefix_unify([]).
 (b)  prefix_unify([S=T|G]) :-
 (c)       ( -S2=S -> T2=T ; -S2=T, T2=S ), flatten(S2,S1), flatten(T2,T1),
 (d)       ( logic(s5) -> tuni_s5(S1,T1) ;
 (e)         logic(d)  -> tuni_d(S1,T1) ;
 (f)         logic(t)  -> tuni_t(S1,[],T1) ;
 (g)         logic(s4) -> tunify(S1,[],T1) ),
 (h)       prefix_unify(G).

(R1)  tuni_t([],[],[]).
(R2)  tuni_t([],[],[X|T])      :- tuni_t([X|T],[],[]).
(R3)  tuni_t([V|S],[],[])      :- V=[], tuni_t(S,[],[]).
(R4)  tuni_t([X1|S],[],[X2|T]) :- (var(X1) -> (var(X2), X1==X2);
                                   (\+var(X2), unify_with_occurs_check(X1,X2))),
                                   !, tuni_t(S,[],T).
(R5)  tuni_t([V|S],[],[X|T])   :- var(V), tuni_t([V|S],[X],T).
(R6)  tuni_t([C|S],[],[V|T])   :- \+var(C), var(V), tuni_t([V|T],[C],S).
(R7)  tuni_t([V1|S],[],[V2|T]) :- var(V1), V2=[], tuni_t(T,[V1],S).
(R8)  tuni_t([V|S],[X],T)      :- V=[], tuni_t(S,[X],T).
(R9)  tuni_t([V|S],[X],T)      :- var(V), unify_with_occurs_check(V,X),
                                   tuni_t(S,[],T).
(R10) tuni_t([C|S],[V],T)      :- \+var(C), var(V), unify_with_occurs_check(V,C),
                                   tuni_t(S,[],T).
```

Figure 8: Source code of the prefix unification for the modal logic T

The predicate `prefix_unify(G)` is used to solve a *set* of prefix equations. Its code is shown in the upper part of Figure 8. The set of prefix equations G is presented by a Prolog list of equations of the form `S=T`. In each step one prefix equation is selected (line *b*) and the appropriate prefix unification is invoked depending on the logic specified by the `logic` predicate (line *d* to *g*). The remaining equations are considered afterwards (line *h*). If all equations are solved (line *a*) the predicate `prefix_unify` succeeds. A minus sign in front of a prefix (used for technical reasons) is deleted and prefixes are flattened (as they may contain nested lists) before the unification starts (line *c*).

The predicates `tuni_s5` and `tuni_d` implementing the prefix unification for the modal logics S5 and D, respectively, are straightforward. Details about the predicate `tunify` implementing the prefix unification for the modal logic S4 can be found in [10].

4.2 The Connection Calculus

The implementation of the connection calculus for first-order modal logic, called MleanCoP, is based on leanCoP 2.0, an automated theorem prover for first-order classical logic [11]. To adapt the implementation to the modal connection calculus of Section 3.2 the leanCoP prover is extended by (a) prefixes that are added to literals and collected during the proof search and (b) an additional list that contains term variables together with their prefixes in order to check the domain condition. First, MleanCoP performs a classical proof search. After a proof is found, the prefixes of the literals in each connection are unified and the domain condition is checked. If the prefix unification fails the (classical) search for alternative connections continues via backtracking. The Prolog source code of the MleanCoP core prover is shown in Figure 9. The underlined code was added to leanCoP; no other modifications were done. In ECLiPSe Prolog sound unification has to be switched on with `set_flag(occur_check,on).`

```
(a)   prove(PathLim,Set) :-
(b)       ( \+member(scut,Set) ->
(c)         prove([(-(#)):(-[])],[],PathLim,[],[PreSet,FreeV1],Set) ;
(d)         lit((#):_,FV:C,_) ->
(e)         prove(C,[(-(#)):(-[])],PathLim,[],[PreSet,FreeV],Set),
(f)         append(FreeV,FV,FreeV1) ),
(g)         domain_cond(FreeV1), prefix_unify(PreSet).
(h)   prove(PathLim,Set) :-
(i)       member(comp(Limit),Set), PathLim=Limit -> prove(1,[]) ;
(j)       (member(comp(_),Set);retract(pathlim)) ->
(k)       PathLim1 is PathLim+1, prove(PathLim1,Set).

(1)   prove([],_,_,_,[[],[]],_).
(2)   prove([Lit:Pre|Cla],Path,PathLim,Lem,[PreSet,FreeV],Set) :-
(3)       \+ (member(LitC,[Lit:Pre|Cla]), member(LitP,Path), LitC==LitP),
(4)       (-NegLit=Lit;-Lit=NegLit) ->
(5)          ( member(LitL,Lem), Lit:Pre==LitL, PreSet3=[], FreeV3=[]
(6)            ;
(7)            member(NegL:PreN,Path), unify_with_occurs_check(NegL,NegLit),
(8)            \+ \+ prefix_unify([Pre=PreN]), PreSet3=[Pre=PreN], FreeV3=[]
(9)            ;
(10)           lit(NegLit:PreN,FV:Cla1,Grnd1),
(11)           \+ \+ prefix_unify([Pre=PreN]),
(12)           ( Grnd1=g -> true ; length(Path,K), K<PathLim -> true ;
(13)             \+ pathlim -> assert(pathlim), fail ),
(14)           prove(Cla1,[Lit:Pre|Path],PathLim,Lem,[PreSet1,FreeV1],Set),
(15)           PreSet3=[Pre=PreN|PreSet1], append(FreeV1,FV,FreeV3)
(16)         ),
(17)         ( member(cut,Set) -> ! ; true ),
(18)         prove(Cla,Path,PathLim,[Lit:Pre|Lem],[PreSet2,FreeV2],Set),
(19)         append(PreSet3,PreSet2,PreSet), append(FreeV2,FreeV3,FreeV).
```

Figure 9: Source code of the MleanCoP core prover

The open subgoal C and the active path $Path$ in the modal connection calculus of Figure 5 are represented by the Prolog lists `Cla` and `Path`, respectively. The matrix M is written into Prolog's database in a preprocessing step. For every clause $C \in M$ and for every literal `Lit:Pre`$\in C$ the fact `lit(Lit:Pre,FV:C1,Grnd)` is stored where `FV` is a list of elements of the form `[V,Pre1]` in which `V` is a term variable in C and `Pre1` is its prefix, `C1`$=C\backslash\{$`Lit:Pre`$\}$ and `Grnd` is `g` if C is ground, otherwise `Grnd` is `n`. Atoms are represented by Prolog atoms, term variables by Prolog variables and negation by "−". The substitutions σ_Q and σ_M are stored implicitly by Prolog.

The predicate `prove(Cla,Path,PathLim,Lem,[PreSet,FreeV],Set)` in line 1 to 19 implements the axiom, the reduction rule and the extension rule of the modal connection calculus of Figure 5. `PathLim` is the maximum size of the active path used for iterative deepening, `Lem` is a set of literals used as lemmata, and `Set` is a list of options used to control the proof search (see [12] for details). For modal logic the lists `PreSet` and `FreeV` were added. They contain a list of prefix equations and a list of term variables with their prefixes that are collected during the proof search.

Line 1 implements the axiom, line 4 calculates the complement of the first literal `Lit:Pre` in the list of open subgoals, which is is used for the following reduction step (line 7/8 and 18/19) or extension step (line 10/11, 14/15 and 18/19). In line 7 it is checked whether the active path `Path` contains a literal `NegL:PreN` that unifies with the complement `NegLit:Pre` of the literal `Lit:Pre`. In Line 8 a *weak prefix unification* of the prefix equation `Pre=PreN` is carried out (double negation prevents any variable bindings). Line 18 and 19 continue the proof search for the premise of the reduction rule.

In line 10 the predicate `lit(NegLit:PreN,FV:Cla1,Grnd1)` is used to find a clause that contains the complement of the literal `Lit:Pre`. `FV` is the list of term variables occuring in `Cla1`, `Cla1` is the remaining set of literals of the selected clause. Again, weak prefix unification in line 11 ensures that the prefixes of the new connection are unifiable. In line 14 the proof search for the left premise of the extension rule is invoked with the new open subgoal `Cla1` and the active path `Path` extended by the literal `Lit:Pre`. Afterwards the proof search for the right premise is invoked in line 18. The lists of prefix equations and term variables are collected in line 8, 15 and 19.

The predicate `prove(PathLim,Set)` in line *a* to *k* implements the start rule of the modal connection calculus in Figure 5. When the matrix M is written into Prolog's database the distinct literal `#:[]` is added to all positive clauses. In line *c* the proof search starts with the open subgoal `[(-(#)):(-[])]` and the empty active path `[]`, i.e. only positive clauses are used as start clause. After the classical proof search succeeds the domain condition is checked and the prefixes of each connection are unified. To this end the two predicates `domain_cond` (see the complete source code on the MleanCoP web site) and `prefix_unify` (see Section 4.1) are invoked in line *g*.

Prolog uses a depth-first search strategy resulting in an incomplete proof search. In order to regain completeness MleanCoP performs an iterative deepening on the size of the active path. When the extension rule is applied and the new clause is not ground, i.e. it contains a (term or prefix) variable, it is checked whether the size `K` of the active path exceeds the current path limit `PathLim` (line 12). In this case the predicate `pathlim` is written into Prolog's database (line 13) indicating the need to increase the path limit. If the proof search for the current path limit fails and the predicate `pathlim` was written into the database (line *j*), then `PathLim` is increased and the proof search starts again (line *k*).

leanCoP uses a few additional techniques, which are integrated into MleanCoP as well; see [12] for details. *Regularity* ensures that no literal occurs more than once in the active path. It is integrated into the modal connection calculus in Figure 5 by imposing the following restriction on the reduction and the extension rule: $\forall\ L'{:}p' \in C \cup \{L_1{:}p_1\} :\ \sigma_Q(\sigma_M(L'{:}p')) \notin \sigma_Q(\sigma_M(Path))$. Line 3 implements a ground version of the regularity condition in MleanCoP. The idea of *Lemmata* or factorization is to reuse subproofs during the proof search. To this end an additional argument `Lem` is added to the `prove` predicate and line 5 is added to the MleanCoP source code. *Restricted backtracking* [12] is a very effective technique to restrict backtracking in the connection calculus. It is implemented in line *b*, *d*, *e*, *f*, and 17. A *definition clausal form* translation reduces the number of possible connections and, hence, the search space. Furthermore, MleanCoP uses a *fixed strategy scheduling*, i.e. the core prover is invoked repeatedly by a shell script with different prover options given in the list `Set`.

Example 10 (MleanCoP). *Consider the modal formula F_1 and its matrix M_1 of Example 4. The goal* `prove(((* ex X:p(X) , # all Y:(* p(Y) => q(Y)) ) => * ex Z:q(Z)))` *succeeds; hence, F_1 is valid. The (internal) representation of M_1 (with added variable/prefix lists) is* `[[]:[-(p(d^[]^[a1])):-([a1])], [[Y,[V1]]]: [p(Y): [V1,V2], -(q(Y)): -([V1])], [[Z,[V3]]]: [q(Z): [V3]]].`

5 Performance

The MleanCoP implementation of Section 4.2 was tested on all 580 uni-modal problems of version 1.1 of the QMLTP library [14]. All problems were converted into the MleanCoP syntax using the TPTP2X tool [15] together with the format file included in the QMLTP library. All tests were conducted on a 3.4 GHz Xeon system with 4 GB of RAM running Linux 2.6.24 and ECLiPSe Prolog 5.10. The CPU time limit for all proof attempts was 600 seconds. Table 3 gives a summary of the test result. It shows the number of solved problems for the varying, cumulative, and constant domain of the modal logics D, T, S4, and S5 for the provers MleanCoP 1.2 (CoP), MleanTAP 1.3 (TAP), and MleanSeP 1.2 (SeP).[5]

Table 3: Number of solved problems from the QMLTP library v1.1

	varying domain			cumulative domain			constant domain		
logic	CoP	TAP	SeP	CoP	TAP	SeP	CoP	TAP	SeP
D	422	104	–	424	124	134	426	139	135
T	369	142	–	374	164	167	381	179	167
S4	393	173	–	432	209	201	434	224	198
S5	447	223	–	479	276	–	479	276	–

MleanSeP implements the standard modal sequent calculus (see Definition 1) for several modal logics. Proof search is carried out in an analytic way and free term variables are used together with a dynamic Skolemization to ensure the Eigenvariable condition. For the constant domain variants the *Barcan formula (scheme)* is added in a preprocessing step. The prover does not deal with the modal logic S5 and the varying domain variants as they do not have (cut-free) sequent calculi [17].

MleanTAP is a compact implementation of a prefixed tableau calculus for several modal logics. Similar to the modal connection calculus presented in Section 3.2 it is based on the modal matrix characterization of validity given in Theorem 1. Like MleanCoP it first performs a purely classical proof search. After a classical tableau proof is found the prefixes of those literals that closed the tableau branches are unified. The existence of a unifier ensures that the given formula is valid in modal logic. MleanCoP and MleanTAP use the same source code for the prefix unification and for checking the domain condition.

Table 4 presents detailed performance results of the three theorem provers on the QMLTP library. It shows for each considered modal logic and domain the number of solved problems, the number of proved (i.e. valid) problems, the number of refuted (i.e. invalid) problems, and the number of problems that are proved within specific time intervals. In general, MleanCoP proves and refutes significantly more problems than MleanTAP and MleanSeP. The number of proved problems by MleanTAP and MleanSeP is similar, but MleanSeP has a better time complexity behaviour; MleanTAP solves only two problems (for S4) in a time of more than 10 seconds. As MleanCoP uses a fixed strategy scheduling, many problems are solved in a time of more than one second. In general, the fewest problems are proved for the modal logic D with varying domain and the highest problems are proved for S5 with cumulative or constant domain. This behaviour is in line with the following fact:

If $\mathit{Theorem}(\mathcal{L},\mathcal{D})$ is the set of theorems in the modal logic $\mathcal{L}$ with the domain $\mathcal{D}$, then for all logics $\mathcal{L} \in \{\text{D,T,S4}\}$ and all domains $\mathcal{D}$ it is: $\mathit{Theorem}(\text{D},\mathcal{D}) \subset \mathit{Theorem}(\text{T},\mathcal{D}) \subset \mathit{Theorem}(\text{S4},\mathcal{D}) \subset \mathit{Theorem}(\text{S5},\mathcal{D})$ and $\mathit{Theorem}(\mathcal{L},\text{varying}) \subset \mathit{Theorem}(\mathcal{L},\text{cumulative}) \subset \mathit{Theorem}(\mathcal{L},\text{constant})$; furthermore, it is $\mathit{Theorem}(\text{S5},\text{varying}) \subset \mathit{Theorem}(\text{S5},\text{cumulative}) = \mathit{Theorem}(\text{S5},\text{constant})$.

[5] None of these provers were tuned towards the problems in the QMLTP library. MleanSeP and MleanTAP can be obtained at `http://www.leancop.de/mleansep/` and `http://www.leancop.de/mleantap/`.

Table 4: Detailed performance results for the modal logics D, T, S4, and S5

logic		varying domain			cumulative domain			constant domain		
		CoP	TAP	SeP	CoP	TAP	SeP	CoP	TAP	SeP
D	solved	422	104	–	424	124	134	426	139	135
	proved	179	100	–	200	120	130	217	135	134
	refuted	243	4	–	224	4	4	209	4	1
	proved 0s to 1s	149	97	–	169	117	126	187	132	101
	1s to 10s	11	3	–	12	3	4	11	3	28
	10s to 100s	7	0	–	7	0	0	7	0	3
	100s to 600s	12	0	–	12	0	0	12	0	2
T	solved	369	142	–	374	164	167	381	179	167
	proved	224	138	–	249	160	163	269	175	166
	refuted	145	4	–	125	4	4	112	4	1
	proved 0s to 1s	199	133	–	223	157	156	238	172	148
	1s to 10s	10	5	–	11	3	6	12	3	10
	10s to 100s	9	0	–	9	0	1	10	0	7
	100s to 600s	6	0	–	6	0	0	9	0	1
S4	solved	393	173	–	432	209	201	434	224	198
	proved	274	169	–	338	205	197	352	220	197
	refuted	119	4	–	94	4	4	82	4	1
	proved 0s to 1s	245	167	–	286	202	152	301	217	146
	1s to 10s	11	2	–	18	2	32	18	2	13
	10s to 100s	11	0	–	21	1	6	18	1	30
	100s to 600s	7	0	–	13	0	7	15	0	8
S5	solved	447	223	–	479	276	–	479	276	–
	proved	359	219	–	438	272	–	438	272	–
	refuted	88	4	–	41	4	–	41	4	–
	proved 0s to 1s	307	215	–	372	269	–	372	269	–
	1s to 10s	23	4	–	24	3	–	24	3	–
	10s to 100s	16	0	–	25	0	–	25	0	–
	100s to 600s	33	0	–	17	0	–	17	0	–

6 Conclusion

An implementation of a modal connection calculus for various first-order modal logics was presented. The modal connection calculus extends the classical calculus by prefixes and an additional prefix unification procedure. The specific modal logic and domain variant is selected through different prefix unification algorithms and different domain conditions, respectively. The Skolemization technique is extended to prefix variables. This makes an explicit reflexivity test of the reduction ordering redundant and simplifies the implementation. The implementation itself is based on the classical leanCoP prover. Optimization techniques used in leanCoP, such as regularity, lemmata, restricted backtracking and a definitional clausal form translation, are adapted and integrated into the modal prover MleanCoP as well.

Experimental results on the QMLTP library indicate that the performance of MleanCoP is significantly higher than the performance of provers based on analytic sequent calculi or prefixed tableau calculi. Hence, a connection-driven strategy seems to be essential for an efficient proof search based on

approaches that use prefixes and a prefix unification procedure. A similar effect was already observed for first-order intuitionistic logic [10].

The development of automated theorem proving systems for first-order modal logic is still in its early stages. Only few other implementations exist so far. Another promising approach is the embedding of first-order modal logic into simple type theory [2, 3]. Future work includes the comparison with implementations of this and other approaches.

While a clausal form technically simplifies the proof search procedure, the translation into clausal form has a negative effect on the number of (possible) connections and, thus, on the size of the resulting search space. A non-clausal connection calculus [13] avoids any translation steps and preserves the structure of the original formula. Future work also includes the improvement of the prefix unification algorithm as this is the component that captures the main part of the additional search required for (first-order) modal logics.

Acknowledgements. The author would like to thank Thomas Raths for testing all implementations on the QMLTP library and providing the presented performance statistics.

References

[1] G. Beckert, R. Goré. Free Variable Tableaux for Propositional Modal Logics. In D. Galmiche, Ed., *TABLEAUX 1997*, LNAI 1227, pp. 91–106. Springer, Heidelberg, 1997.

[2] C. Benzmüller, L. Paulson. Quantified Multimodal Logics in Simple Type Theory. SEKI Report SR–2009–02 (ISSN 1437–4447), Saarland University, 2009.

[3] C. Benzmüller, L. Paulson. Multimodal and Intuitionistic Logics in Simple Type Theory. *The Logic Journal of the IGPL*, 18(6):881–892, 2010.

[4] W. Bibel. *Automated Theorem Proving*. Vieweg, Wiesbaden, 1987.

[5] M. Fitting. *Proof Methods for Modal and Intuitionistic Logic*. D. Reidel, Dordrecht, 1983.

[6] M. Fitting, R. L. Mendelsohn. *First-Order Modal Logic*. Kluwer, Dordrecht, 1998.

[7] G. Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift* 39:176–210, 405–431, 1935.

[8] R. Hähnle. Tableaux and Related Methods. In A. Robinson, A. Voronkov, Eds., *Handbook of Automated Reasoning*, pp. 100–178. Elsevier, Amsterdam, 2001.

[9] U. Hustadt, R. A. Schmidt. MSPASS: Modal Reasoning by Translation and First-Order Resolution. R. Dyckhoff, Ed., *TABLEAUX 2000*, LNAI 1847, pp. 67–81. Springer, Heidelberg, 2000.

[10] J. Otten. Clausal Connection-Based Theorem Proving in Intuitionistic First-Order Logic. In B. Beckert, Ed., *TABLEAUX 2005*, LNAI 3702, pp. 245–261. Springer, Heidelberg, 2005.

[11] J. Otten. leanCoP 2.0 and ileanCoP 1.2: High Performance Lean Theorem Proving in Classical and Intuitionistic Logic. In A. Armando, P. Baumgartner, G. Dowek, Eds., *IJCAR 2008*, LNCS 5195, pp. 283–291. Springer, Heidelberg, 2008.

[12] J. Otten. Restricting Backtracking in Connection Calculi. *AI Communications* 23:159–182, 2010.

[13] J. Otten. A Non-clausal Connection Calculus. In K. Brünnler, G. Metcalfe, Eds., *TABLEAUX 2011*, LNAI 6793, pp. 226–241. Springer, Heidelberg, 2011.

[14] T. Raths, J. Otten. The QMLTP Problem Library for First-order Modal Logics. 2012. *Submitted*.

[15] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[16] A. Waaler. Connections in Nonclassical Logics. In A. Robinson, A. Voronkov, Eds., *Handbook of Automated Reasoning*, pp. 1487–1578. Elsevier, Amsterdam, 2001.

[17] L. Wallen. *Automated deduction in nonclassical logic*. MIT Press, Cambridge, 1990.