The Modal Logic Programming System MProlog *

Linh Anh Nguyen

Institute of Informatics, University of Warsaw
ul. Banacha 2, 02-097 Warsaw, Poland
nguyen@mimuw.edu.pl

Abstract. We present the design of our implemented modal logic programming system MProlog. This system is written in Prolog as a module for Prolog. Codes, libraries, and most features of Prolog can be used in MProlog programs. The system contains a number of built-in SLD-resolution calculi for modal logics, including calculi for useful multimodal logics of belief. It is a tool to experiment with applications of modal logic programming to AI.

1 Introduction

Modal logics can be used to reason about knowledge, belief, actions, etc. Many authors have proposed modal extensions for logic programming [4, 2, 1, 5, 12, 3, 9, 10]. There are two approaches: the direct approach and the translational approach. The first approach directly uses modalities, while the second one translates modal logic programs to classical logic programs. The works by Akama [1], Debart et al [5], and Nonnengart [12] use the translational approach, while the works by Fariñas del Cerro [4], Balbiani et al [2], Baldoni et al [3], and our works [9, 10] use the direct approach.

In modal logic programming, it seems more intuitive and convenient to use modalities in a direct way[1] (e.g., in the debugging and interactive modes of programming). In the direct approach, the work by Balbiani et al [2] disallows $\Box$ in bodies of program clauses and goals, and the work by Baldoni et al [3] disallows $\Diamond$ in programs and goals. In the Molog system implemented by the group of Fariñas del Cerro [4], universal modal operators in bodies of program clauses and goals are also translated away.

In [9], we developed a fixpoint semantics, the least model semantics, and an SLD-resolution calculus in a direct way for modal logic programs in all of the basic serial[2] monomodal logics. There are two important properties of our approach in [9]: no special restriction on occurrences of $\Box$ and $\Diamond$ is required (programs

* In J.J. Alferes and J.A. Leite (Eds.): Logics in Artificial Intelligence, 9th European Conference, JELIA 2004, Lisbon, Portugal, September 27-30, 2004, Proceedings. LNCS 3229, pages 266–278, Springer, 2004.

[1] Somehow one still has to use skolemization or a labeling technique for $\Diamond$.

[2] A monomodal logic is serial if it contains the axiom $\Box\varphi \rightarrow \Diamond\varphi$.

and goals are of a normal form but the language is as expressive as the general modal Horn fragment) and the semantics are formulated closely to the style of classical logic programming (as in Lloyd's book [8]). In [10], we generalized the methods and extended the results of [9] for multimodal logic programming, giving a general framework for developing semantics of multimodal logic programs and presenting sound and complete SLD-resolution calculi for a number of useful multimodal logics of belief, which will be considered in this work.

Despite that the theory of modal logic programming has been studied in a considerable number of works, it has not received much attention in practice. But if we want to use modal logics for practical applications, then modal logic programming deserves for further investigations, especially in practical issues.

As far as we know, amongst the works by other authors that use the direct approach for modal logic programming, only the Molog system proposed by Fariñas del Cerro [4] has been implemented. The current version of this system has, however, some drawbacks in design. It only says yes or no without giving computed answers. Molog uses a special predicate, named *prolog*, to call formulas of Prolog, which is undesirable when the amount of Prolog code is not small. Molog uses <-- instead of :-, and & instead of ',', which means that Prolog program files cannot be included in Molog programs.

In this work, we present the design of our implemented modal logic programming system MProlog [11], whose theoretical foundations are our work [9, 10]. Our system is written in Prolog as a module for Prolog. Codes, libraries, and most features of Prolog can be used in MProlog programs in a pure way. The system contains a number of built-in SLD-resolution calculi for modal logics, including calculi for multimodal logics intended for reasoning about multi-degree belief, for use in distributed systems of belief, or for reasoning about epistemic states of agents in multi-agent systems.

Due to the lack of space, we will not discuss implementation details. The design of MProlog presented here together with comments given in code files [11] is sufficient to understand the implementation of the system. We assume that the reader is familiar with the classical SLD-resolution calculus and Prolog.

2 Preliminaries

2.1 Syntax and Semantics of Quantified Multimodal Logics

A language for quantified multimodal logics is an extension of the language of classical predicate logic with modal operators $\Box_i$ and $\Diamond_i$, for $1 \leq i \leq m$ (where m is fixed). If $m = 1$ then we ignore the subscript i and write $\Box$ and $\Diamond$. The operators $\Box_i$ are called universal modal operators, while $\Diamond_i$ are called existential modal operators. Terms and formulas are defined in the usual way, with an emphasis that if φ is a formula then $\Box_i\varphi$ and $\Diamond_i\varphi$ are also formulas.

A *Kripke frame* is a tuple $\langle W, \tau, R_1, \ldots, R_m\rangle$, where W is a nonempty set of possible worlds, $\tau \in W$ is the *actual world*, and R_i is a binary relation on W, called the *accessibility relation* for the modal operators $\Box_i$, $\Diamond_i$. If $R_i(w, u)$ holds then we say that the world u is accessible from the world w via R_i.

A *fixed-domain Kripke model with rigid terms*, hereafter simply called a Kripke model or just a model, is a tuple $M = \langle D, W, \tau, R_1, \ldots, R_m, \pi \rangle$, where D is a set called the *domain*, $\langle W, \tau, R_1, \ldots, R_m \rangle$ is a Kripke frame, and π is an interpretation of constant symbols, function symbols and predicate symbols. For a constant symbol a, $\pi(a)$ is an element of D. For an n-ary function symbol f, $\pi(f)$ is a function from D^n to D. For an n-ary predicate symbol p and a world $w \in W$, $\pi(w)(p)$ is an n-ary relation on D.

A *variable assignment* V w.r.t. a Kripke model M is a function that maps each variable to an element of the domain of M. The value of $t^M[V]$ for a term t is defined as usual.

Given some Kripke model $M = \langle D, W, \tau, R_1, \ldots, R_m, \pi \rangle$, some variable assignment V, and some world $w \in W$, the *satisfaction relation* $M, V, w \vDash \psi$ for a formula ψ is defined as follows:

$$
\begin{array}{lll}
M, V, w \vDash p(t_1, \ldots, t_n) & \text{iff} & (t_1^M[V], \ldots, t_n^M[V]) \in \pi(w)(p); \\
M, V, w \vDash \Box_i \varphi & \text{iff} & \text{for all } v \in W \text{ such that } R_i(w, v),\ M, V, v \vDash \varphi; \\
M, V, w \vDash \forall x.\varphi & \text{iff} & \text{for all } a \in D,\ (M, V', w \vDash \varphi), \\
 & & \text{where } V'(x) = a \text{ and } V'(y) = V(y) \text{ for } y \neq x;
\end{array}
$$

and as usual for other cases (treating $\Diamond_i \varphi$ as $\neg \Box_i \neg \varphi$, and $\exists x.\varphi$ as $\neg \forall x. \neg \varphi$). We say that M satisfies φ, or φ is true in M, and write $M \vDash \varphi$, if $M, V, \tau \vDash \varphi$ for every V. For a set Γ of formulas, we call M a model of Γ and write $M \vDash \Gamma$ if $M \vDash \varphi$ for every $\varphi \in \Gamma$.

2.2 Modal Logics of Belief

If as the class of admissible interpretations we take the class of all Kripke models (with no restrictions on the accessibility relations) then we obtain a quantified multimodal logic which has a standard Hilbert-style axiomatization denoted by $K_{(m)}$. Normal modal logics are extensions of $K_{(m)}$ using additional axioms. A modal logic L is *serial* if it contains the axiom $\Box_i \varphi \rightarrow \neg \Box_i \neg \varphi$ (for all $1 \leq i \leq m$). To reflect properties of belief, one can extend $K_{(m)}$ with some of the following axioms (see [10] for the corresponding restrictions on the accessibility relations and [6, 7] for further readings on first-order modal logics):

Name	Schema	Meaning
(D)	$\Box_i \varphi \rightarrow \neg \Box_i \neg \varphi$	belief is consistent
(I)	$\Box_i \varphi \rightarrow \Box_j \varphi$ if $i > j$	subscript indicates degree of belief
(4)	$\Box_i \varphi \rightarrow \Box_i \Box_i \varphi$	belief satisfies positive introspection
(4_s)	$\Box_i \varphi \rightarrow \Box_j \Box_i \varphi$	belief satisfies strong positive introspection
(5)	$\neg \Box_i \varphi \rightarrow \Box_i \neg \Box_i \varphi$	belief satisfies negative introspection
(5_s)	$\neg \Box_i \varphi \rightarrow \Box_j \neg \Box_i \varphi$	belief satisfies strong negative introspection

By adding appropriate combinations of these axioms to $K_{(m)}$, we obtain the modal logics $KDI4$, $KDI4_s$, $KDI45$, $KDI4_s5$ for reasoning about multi-degree belief, the modal logic $KD4_s5_s$ for use in distributed systems of belief, and the modal logic $KD45_{(m)}$ for reasoning about epistemic states of agents in

multi-agent systems. (We use a subscript in $KD45_{(m)}$ to distinguish it from the monomodal logic $KD45$.) In the mentioned modal logics of multi-degree belief, the axiom (I) gives $\Box_i\varphi$ the meaning "φ is believed up to degree i"; while in the logics $KD4_s5_s$ and $KD45_{(m)}$, the formula $\Box_i\varphi$ means "the agent i believes in φ". For reasoning about belief and common belief of groups of agents, there is a multimodal logic denoted by $KD4I_g5_a$ which combines features of $KD45_{(m)}$ and $KDI4$. For further descriptions of the mentioned logics, see [10].

2.3 Modal Logic Programs

In this subsection, we define a modal logic programming language called MProlog, which is a purely logical formalism. Its implementation as an extension of Prolog has a specific syntax and will be studied in Section 4.

A *modality* is a (possibly empty) sequence of modal operators. A *universal modality* is a modality which contains only universal modal operators. We use $\triangle$ to denote a modality and $\boxdot$ to denote a universal modality. Similarly as in classical logic programming, we use a clausal form $\boxdot(\varphi \leftarrow \psi_1, \ldots, \psi_n)$ to denote the formula $\forall(\boxdot(\varphi \vee \neg\psi_1 \ldots \vee \neg\psi_n))$. We use E to denote a classical atom and A, B_1, ..., B_n to denote formulas of the form E, $\Box_i E$, or $\Diamond_i E$.

A *program clause* is a formula of the form $\boxdot(A \leftarrow B_1, \ldots, B_n)$, where $\boxdot$ is a universal modality and $n \geq 0$. $\boxdot$ is called the *modal context*, A the *head*, and $B_1, \ldots, B_n$ the *body* of the program clause.

An *MProlog program* is a finite set of program clauses. An *MProlog goal atom* is a formula of the form $\boxdot E$ or $\boxdot\Diamond_i E$, where $\boxdot$ is a universal modality. An *MProlog goal* is a formula written in the clausal form $\leftarrow \alpha_1, \ldots, \alpha_k$, where each α_i is an MProlog goal atom.

It is shown in [10] that MProlog has the same expressiveness power as the general Horn fragment in normal modal logics. For a specific logic L, we may adopt some restrictions on modal contexts of MProlog program clauses and MProlog goals and call the obtained language *L-MProlog*. Such restrictions either follow from equivalencies in L or are acceptable from the practical of view, and furthermore, they do not reduce expressiveness of the language.

For example, in $KDI4_s5$ we have the equivalence $\nabla\nabla'\varphi \equiv \nabla'\varphi$, where ∇ and ∇' are modal operators. Hence we can assume that the modal context of an $KDI4_s5$-*MProlog program clause* has length 1 or 0, and an $KDI4_s5$-*MProlog goal* has goal atoms of the form E, $\Box_i E$, or $\Diamond_i E$ with E being a classical atom. See [10] for restrictions of L-MProlog in the other modal logics of belief.

3 A Framework of SLD-Resolution for MProlog

In [10], we give a general framework for developing fixpoint semantics, least model semantics, and SLD-resolution calculi for L-MProlog, where L is a serial modal logic whose frame restrictions, except seriality, are Horn clauses (in particular, L can be any one of the modal logics of belief considered in this paper). In this section, we outline the fragment involving SLD-resolution of that framework. For fixpoint semantics, the reader is referred to [10].

A modal operator is now $\Box_i$, $\Diamond_j$, or $\langle S\rangle_k$, where $\langle S\rangle_k$ is a $\Diamond_k$ labeled by S which is either a classical atom or a variable for classical atoms (called an *atom variable*). For further information on labeled modal operators, see [10].

We use ∇ and ∇' to denote modal operators, $\triangle$ to denote a modality (which now may contain labeled modal operators). A *modal atom* is a formula of the form $\triangle E$. A *simple modal atom* is a formula of the form E or ∇E. We use A, B to denote simple modal atoms, and α, β to denote modal atoms.

There may exist a compact form for modalities in L. For each specific modal logic L, we define the *L-normal form of modalities*. For example, a modality is in $KDI4_s5$-normal form if its length is 0 or 1. It is possible that no restriction is adopted for L-normal form of modalities. A modality is in *L-normal labeled form* if it is in L-normal form and does not contain unlabeled existential modal operators $\Diamond_i$. A modal atom $\triangle E$ is in L-normal (labeled) form if $\triangle$ is in L-normal (labeled) form. Given a ground modal atom not in L-normal form, the NF_L *operator* converts it to L-normal form.

Given a modal atom α, one can derive other modal atoms from α using axioms of L. The corresponding operator is called the Sat_L *operator*. The "direct consequences" operator $T_{L,P}$ is defined using Sat_L and NF_L. An SLD-resolution calculus can be viewed as a reversed analogue of a direct consequences operator. Hence, to define an SLD-resolution calculus for L-MProlog we need reversed analogues of the operators Sat_L and NF_L. These operators are called the $rSat_L$ *operator* and the rNF_L *operator*, respectively. See [10, 9] for formal definitions of the operators Sat_L, NF_L, $rSat_L$, and rNF_L.

The $rSat_L/rNF_L$ operators are each specified by a finite set of rules of the form $\alpha \leftarrow \beta$, where α and β are (schemata of) modal atoms. The rules are used as meta-clauses (i.e. schemata of clauses) in SLD-derivations. Such rules can be accompanied by conditions which specify when the rule can be used.

As an example, for $L = KDI4_s5$, the rNF_L operator is specified by the only rule $\nabla E \leftarrow \langle X\rangle_i \nabla E$, and the $rSat_L$ operator is specified by three rules: (a) $\nabla\nabla' E \leftarrow \nabla' E$, (b) $\Diamond_i E \leftarrow \Diamond_j E$ if $i > j$, (c) $\Diamond_i E \leftarrow \langle X\rangle_i E$; where X is a fresh[3] atom variable. We will use these rules for the example in the next page.

Resolvents of a goal $\leftarrow \alpha_1, \ldots, \alpha_k$ and an $rSat_L/rNF_L$ rule $\alpha \leftarrow \beta$ are defined in the usual way. For example, resolving $\leftarrow \Box_1\Diamond_2 p(x)$ with the rule $\nabla\nabla' E \leftarrow \nabla' E$ results in $\leftarrow \Diamond_2 p(x)$, since ∇ is instantiated to $\Box_1$, and ∇' is instantiated to $\Diamond_2$.

For each specific modal logic L, we define a pre-order $\preceq_L$ to compare modal operators. For example, for $L = KDI4_s5$, the pre-order $\preceq_L$ is the least reflexive and transitive binary relation between modal operators such that: $\Diamond_i \preceq_L \langle S\rangle_i \preceq_L \Box_i$, and if $i < j$ then $\Box_i \preceq_L \Box_j$ and $\Diamond_j \preceq_L \Diamond_i$. If $\nabla \preceq_L \nabla'$ then we say that ∇ is an *L-instance* of ∇'. We say that an atom $\triangle E$ is an *L-instance* of $\triangle' E'$ if $\triangle$ and $\triangle'$ have the same length k and there exists a substitution θ such that $E = E'\theta$ and for any $1 \le i \le k$, the modal operator in position i of $\triangle$ is an L-instance of the modal operator in position i of $\triangle'\theta$.

[3] This means that *standardizing* is also needed for atom variables.

The *forward labeled form* of an atom α is the atom α' such that if α is of the form $\triangle\Diamond_i E$ then $\alpha' = \triangle\langle E\rangle_i E$, else $\alpha' = \alpha$. For example, the forward labeled form of $\Diamond_1 s(a)$ is $\langle s(a)\rangle_1 s(a)$.

If $\boxdot$ and $\boxdot'$ are universal modalities, and furthermore, $\boxdot$ is a modal context of an L-MProlog program clause, then we say that $\boxdot'$ is an *L-context instance* of $\boxdot$ if $\boxdot\varphi \rightarrow \boxdot'\varphi$ is a theorem in L for an arbitrary φ. For example, $\Box_1$ is a $KDI4_s5$-context instance of $\Box_2$.

Let $G = \leftarrow \alpha_1, \ldots, \alpha_i, \ldots, \alpha_k$ be a goal and $\varphi = \boxdot(A \leftarrow B_1, \ldots, B_n)$ a program clause. Then G' is *derived* from G and φ in L using mgu θ, and called an *L-resolvent* of G and φ, if the following conditions hold:

- $\alpha_i = \triangle' A'$, with $\triangle'$ in L-normal labeled form, is called the *selected atom*.
- $\triangle'$ is an L-instance of $\boxdot'$ which is an L-context instance of $\boxdot$.
- θ is an mgu such that: $A'\theta$ has the same classical atom as $A\theta$, and $A'\theta$ is an L-instance of the forward labeled form of $A\theta$.
- G' is the goal $\leftarrow (\alpha_1, \ldots, \alpha_{i-1}, \triangle' B_1, \ldots, \triangle' B_n, \alpha_{i+1}, \ldots, \alpha_k)\theta$.

For example, the unique $KDI4_s5$-resolvent of $\leftarrow \Box_1 p(x)$ and $\Box_2(p(x) \leftarrow \Diamond_2 q(x))$ is $\leftarrow \Box_1\Diamond_2 q(x)$ (here, $\boxdot = \Box_2$ and $\triangle' = \boxdot' = \Box_1$). As another example, the unique $KDI4_s5$-resolvent of $\leftarrow \langle Y\rangle_1\langle X\rangle_1 r(x), \langle X\rangle_1 s(x)$ and $\Box_1(\Box_1 r(x) \leftarrow s(x))$ is $\leftarrow \langle Y\rangle_1 s(x), \langle X\rangle_1 s(x)$ (here, $\boxdot = \Box_1$ and $\triangle' = \boxdot' = \langle Y\rangle_1$).

SLD-derivation is defined using two kinds of steps: a) resolving a goal with a program clause, b) resolving a goal with an $rSat_L/rNF_L$ rule. SLD-refutation and computed answer are defined in the usual way.

Using the framework, in [10] we have given sound and complete SLD-resolution calculi for L-MProlog for all the modal logics of belief considered in this work.

As an example, consider the goal $G = \leftarrow \Box_1 p(x)$ and the program P:

$$\begin{array}{l}\varphi_1 = \Box_2(p(x) \leftarrow \Diamond_2 q(x))\\ \varphi_2 = \Box_1(q(x) \leftarrow r(x), s(x))\\ \varphi_3 = \Box_1(\Box_1 r(x) \leftarrow s(x))\\ \varphi_4 = \Diamond_1 s(a) \leftarrow\end{array}$$

Here is an SLD-refutation of $P \cup \{G\}$ in $L = KDI4_s5$:

Goals	Input clauses/rules	MGUs
$\leftarrow \Box_1 p(x)$		
$\leftarrow \Box_1\Diamond_2 q(x)$	φ_1	$\{x_1/x\}$
$\leftarrow \Diamond_2 q(x)$	$rSat_L(a)$	
$\leftarrow \Diamond_1 q(x)$	$rSat_L(b)$	
$\leftarrow \langle X\rangle_1 q(x)$	$rSat_L(c)$	
$\leftarrow \langle X\rangle_1 r(x), \langle X\rangle_1 s(x)$	φ_2	$\{x_5/x\}$
$\leftarrow \langle Y\rangle_1\langle X\rangle_1 r(x), \langle X\rangle_1 s(x)$	rNF_L	
$\leftarrow \langle Y\rangle_1 s(x), \langle X\rangle_1 s(x)$	φ_3	$\{x_7/x\}$
$\leftarrow \langle X\rangle_1 s(a)$	φ_4	$\{x/a, Y/s(a)\}$
empty clause	φ_4	$\{X/s(a)\}$

4 Design of MProlog

Starting from the purely logical formalism of MProlog, we have built a real system for it [11]. The implemented system adds extra features to the purely logical formalism in order to increase usefulness of the language. It is written in Prolog and can run in SICStus Prolog and SWI-Prolog. From now on we use MProlog to refer to the implemented system.

MProlog is designed as an extension of Prolog. This means that we can use Prolog codes, libraries and most features of Prolog in MProlog programs. This gives MProlog capabilities for real applications. MProlog is implemented as a module for Prolog and does not have its own running environment. It provides instead a list of built-in predicates to be used in Prolog.

4.1 Syntax of MProlog Programs

In MProlog, there are three kinds of predicates: classical predicates, modal predicates, and classical predicates which are defined using modal formulas. Predicates of the last kind are called *dum* predicates. The semantics of classical predicates and *dum* predicates does not depend on worlds in Kripke models. If E is a classical atom of a classical predicate or a *dum* predicate then $\triangle E \equiv E$ for every modality $\triangle$. An MProlog program consists of *modal fragments* and *classical fragments* (in an arbitrary number and order). Predicates defined in classical fragments are *classical predicates*. *Dum predicates* are declared in an MProlog program as follows

:- dum_pred $Pred_1$, ..., $Pred_n$.

where each $Pred_i$ is a pair $Name_i/Arity_i$. *Dum* predicates are defined in modal fragments by clauses of the form *E :- Body*, where E is a classical atom. A predicate defined in a modal fragment and not declared earlier as a *dum* predicate is a *modal predicate*.

From now on, by a *calculus* we mean an SLD-resolution calculus for MProlog. An MProlog program may use different calculi, as explained in Section 5. In an MProlog program, a modal fragment starts with a declaration of the form:

:- calculus Cal_1, ..., Cal_n.

where Cal_1, ..., Cal_n are names of calculi. These calculi are called the *calculi of the fragment*. If an MProlog program is loaded by *mconsult(File,Cal)* then the program in *File* is treated as if it begins with

:- calculus Cal.

A modal fragment ends either by a declaration of another modal fragment, or by the end of the program, or by one of the two following declarations:

:- calculus classical.

:- end.

In MProlog, modalities are represented as lists, e.g., as follows:

$\Box\Diamond q(x,y)$	$[b,d] : q(X,Y)$
$\Box_i\langle X\rangle_3\Diamond_j q(a)$	$[bel(I), pos(3,X), pos(J)] : q(a)$
$\Box_x god\_exists \leftarrow christian(x)$	$[bel(X)] : god\_exists$:- $christian(X)$

Here, b stands for "box", d for "diamond", *bel* for "believes", and *pos* for "possible". We use $\triangle : \varphi$ to represent $\triangle\varphi$. Notations of modal operators depend on how the base SLD-resolution calculus is defined. As another example, for MProlog-$\Box$ [10], which disallows existential modal operators in program clauses and goals, we represent $\Box_{i_1} \ldots \Box_{i_k}$ as $[I1, \ldots, Ik]$ (see *belief_box.cal* of [11]).

Syntactically, an MProlog program is a Prolog program. Modal fragments in an MProlog program may contain directives and clauses. Each clause in a modal fragment is of one of the following forms:

Context : (Head :- Body).
Head :- Body.

where *Context* is a list representing a modality, *Head* is of the form E or $M : E$, where E is a classical atom (in the sense of Prolog) and M is a list containing one modal operator. All clauses in a modal fragment are *declared* to all of the calculi of the fragment.

4.2 Syntax of SLD-Resolution Calculi for MProlog

SLD-resolution calculi for MProlog are specified using the framework given in Section 3 and written in Prolog. An SLD-resolution calculus for L contains $rSat_L/rNF_L$ rules, definitions of auxiliary predicates, and definitions for the following required predicates:

1. *universal_modal_operator(Calculus, Operator)*
2. *dual_modal_operator(Calculus, Operator, DualOperator)*
3. *box_lifting_form(Calculus, ModalOperator, BoxLiftingForm)*
   which returns true iff *BoxLiftingForm* is the universal modal operator of the same modal index as *ModalOperator* in the calculus *Calculus*
4. *forward_labeled_form(Calculus, SimpleModalAtom, ForwardLabeledF)*
5. *normal_labeled_form(Calculus, Modality)*
6. *operator_instance(Calculus, Instance, ModalOperator)*
7. *context_instance(Calculus, UniversalModality, ModalContext)*

If the option *check_in_two_steps* of the defined calculus is set to *true*, then instead of the last three predicates of the above list, the calculus must implement two predicates with name prefixed by *pre_check_* or *post_check_* for each of the replaced predicates.

Let us discuss the usefulness of the *check_in_two_steps* option. Suppose that we want to reason about multi-degree belief. We represent, e.g., $\Box_i(p(x) \leftarrow q(x))$ by [bel(I)]: (p(X) :- q(X)). Thus we use variables like I for degrees of belief. Sometimes it is better to delay instantiating such variables to concrete values in order to eliminate branching. If we want to allow users to have ability to turn on/off this option of delaying, then the defined calculus should be designed with the *check_in_two_steps* option turned on. The intention of *pre_check* predicates is to check the involved condition as much as possible without generating branch points and to pass unchecked fragments of the condition to *post_check* predicates, which will be fired latter. In Section 5, we will describe the functioning of those predicates in the MProlog interpreter.

A *resolution cycle* is a derivation consisting of a sequence of applications of $rSat_L/rNF_L$ rules and an application of a program clause. To create an ability to reduce nondeterminism, we provide 4 categories (kinds) of rules: *pre_rSat*, *rSat*, *post_rSat*, and *rNF*. Informally, operators *pre_rSat* and *post_rSat* are deterministic, while operators *rSat* and *rNF* are nondeterministic. This means that when the system tries to resolve a modal atom using an operator of the category *pre_rSat* or *post_rSat*, the first applicable rule of the category will be used, and when the system wants to use *rSat* (resp. *rNF*), different sequences of *rSat* rules (resp. *rNF* rules) will be tried. Lengths of such sequences of *rSat* rules (resp. *rNF* rules) are restricted by the option *limit_rSat* (resp. *limit_rNF*) of the calculus.

Rules of the mentioned categories are of one of the following forms:

AtomIn :- PreCondition, AtomOut, PostComputation.

RuleName :: (AtomIn :- PreCondition, AtomOut, PostComputation).

AtomIn and *AtomOut* are atoms of the form $M : E$, where M (standing for a modality) and E (standing for a classical atom) may be variables in Prolog, and M may be also a list. *RuleName* is a name in Prolog. *PreCondition* and *PostComputation* are (possibly empty) sequences of formulas in Prolog separated by ','. *AtomOut* is called the *atom out* of the rule. It is the first outer (w.r.t. ',') atom of the form $M : E$ of the body of the rule. Names of rules are unique. If a rule is declared without a name, it will be given a unique name by the system. We give below an example rule, a version of : $\Diamond_i E \leftarrow \Diamond_j E$ if $i > j$.

```
rSatKDI4s5 :: ( [pos(I)]:E :-
    get_calling_history(rSat, Cal, _, RNames),
    \+ memberchk(rSatKDI4s5, RNames), % not called before
    pre_compare_deg(Cal, I > J), [pos(J)]:E, post_compare_deg(Cal, I > J)).
```

As shown in the above example, designers of calculi have access to the history of rules called in the current resolution cycle. Such a history for a given category of rules is obtained by

get_calling_history(RuleCategory, Calculus, CalledAtom, RuleNames)

where the last three arguments are outputs. *RuleNames* is the list of names of rules of the category *RuleCategory* which have been applied, in the reverse order, in the current resolution cycle for the beginning atom *CalledAtom* .

Syntactically, rules are clauses in Prolog. They are *defined* as usual clauses. Rules are *declared* to calculi either by a *section of rules* or by a directive. A section of rules is a list of (definitions of) rules bounded by directives. A directive opening a section of rules is of the following form:

:- RuleCategory Cal_1, ..., Cal_n.

where *RuleCategory* is one of *pre_rSat*, *rSat*, *post_rSat*, *rNF*; and $Cal_1, \ldots, Cal_n$ are names of calculi, to which the rules in the section are declared. A section of rules is closed by any directive in Prolog or by the end of the main file. Rules of the same category can also be declared to a calculus using a directive of the following form:

:- set_list_of_mrules(Calculus, RuleCategory, ListOfRuleNames).

Some options are automatically created with default values for each loaded calculus. A definition of a calculus can change values of those options using

set_option(Option, Calculus, Value) and set new options for itself (e.g., a numeric option like *max_modal_index* is needed for modal logics of multi-degree belief).

4.3 Options and Built-in Predicates of MProlog

Before listing built-in predicates of MProlog, we give a list of options of MProlog, which may affect the way the system interprets MProlog programs. There are two kinds of options: options for calculi, and options for the system. Setting values of options is done by the following predicates:

set_option(OptionName, Calculus, Value)
set_option(OptionName, Value)

There are the following built-in options for calculi: *limit_modality_length* (default: 4), *limit_rSat* (default: 3), *limit_rNF* (default: 1), *use_calling_history* (default: false), *check_in_two_steps* (default: false). The *limit_modality_length* option is used to restrict lengths of modalities that may appear in derivations. The options *limit_rSat* and *limit_rNF* have been described in the previous subsection. For some built-in calculi, those numeric limits are firmly set, as they follow from the nature of the base modal logic. In general, they are used to restrict the search space and may affect completeness of the calculus. The boolean option *use_calling_history* should be turned on if rules of the calculus use the history of rules called in the current resolution cycle. The boolean option *check_in_two_steps* has been discussed in the previous subsection.

The boolean option *loop_checking* is a useful option of the system. If it is turned on, the MProlog interpreter will check whether the current modal atom to be resolved has already appeared in the current derivation in order to prevent infinite loops. There are also other options of the system: *loop_checking_stack_size* (default: 300), *random_selection_of_rules* (default: false), *debug* (default: false), *current_calculus* (default: classical), and *priority_list_of_calculi* (default not set).

There are three groups of built-in predicates which are useful for users: main predicates (for consulting, calling, and tracing), predicates for getting and displaying the status of the system, and predicates for dynamic modification of programs. We list here only main predicates of the system[4]:

consult_calculi(Files),
mconsult(ProgramFile, Calculus), *mconsult(ProgramFile)*,
mcall(Goal, Calculus), *mcall(Goal)*,
mtrace, *nomtrace*.

Our MProlog module can be loaded by consulting the file "mprolog.pl" of the package. The user can then load SLD-resolution calculi for modal logics using the predicate *consult_calculi*, whose argument may be a file name or a list of file names. The user can consult MProlog programs using the predicate *mconsult/2* (see Section 4.1 for the meaning of the second argument). *mconsult(ProgramFile)* is treated as *mconsult(ProgramFile, classical)*. Goals involved with modal logics can be asked using the predicate *mcall/2*, where the second argument indicates

[4] See [11] for predicates of the remaining groups.

the calculus in which the goal is asked. If a default calculus is set using the *current_calculus* option, then *mcall/1* can be used instead of *mcall/2*. The predicates *mtrace* and *nomtrace* are used to turn on and off the trace mode for MProlog (which concentrates on modal formulas and is not the trace mode of Prolog).

5 The MProlog Interpreter

The MProlog interpreter is realized by the predicate *mcall(Goal, Calculus)*, which initiates some variables and then calls *mcall_ (Goal, Calculus)*. In this section, we describe in detail the latter predicate, ignoring some aspects like loop checking, updating the history of called rules, or effects of options.

The predicate of the argument *Goal* belongs to one of the following groups: control predicates[5] (\+, ';', ',', ->, if/3), classical predicates, *dum* predicates, and modal predicates.

If *Goal* is an atom of a classical predicate, then *mcall_ (Goal,_)* is defined as *Goal* itself. Formulas *PreCondition* and *PostComputation* from a rule *Head :- PreCondition, AtomOut, PostComputation* are treated as atoms of classical predicates, despite that they may be complicated formulas.

Because *dum* predicates can be defined in different calculi and their semantics does not depend on worlds in Kripke models, they can be used to mix different calculi. If *Goal* is an atom of a *dum* predicate then to resolve *mcall_ (Goal, Calculus)*, the interpreter will try to resolve *Goal* first in *Calculus* and then in different calculi as well. The list of those latter calculi is determined by the value of the *priority_list_of_calculi* option if it is set, and by the list of all calculi otherwise; both cases exclude *Calculus* (the argument). Resolving *Goal* of a *dum* predicate in a calculus *Cal* is done as follows: select a modal clause *Head :- Body* of *Cal*, unify *Goal* with *Head*, and then call *mcall_ (Body, Cal)*.

For the case of modal atoms, we first discuss some auxiliary predicates.

Resolving a modal atom *Goal* with a rule

Head :- PreCondition, AtomOut, PostComputation

is done by unifying *Goal* with *Head*, executing *PreCondition*, and returning *AtomOut* and *PostComputation* as outputs. This task is done by the predicate

solve_using_mrule(Cal, Cat, RName, AtomIn, AtomOut, PostComputation)

with *AtomIn* = *Goal* and *Cal*, *Cat*, *RName* being respectively the calculus, the category, and the name of the rule.

Resolving a modal atom *Goal* using a sequence of rules is done by calling the above described predicate *solve_using_mrule* for each rule of the sequence, where *AtomIn* of the first call of *solve_using_mrule* is *Goal*, and *AtomIn* of each one of the next calls is *AtomOut* of the previous call. As outputs, it returns *AtomOut* of the last call of *solve_using_mrule* and the composition (using ',' and the reverse order) of the obtained *PostComputation* formulas. If the sequence of rules is empty then the outputs are *Goal* and *true*.

To resolve a modal atom *Goal* using rules of a calculus *Cal* that belong to a category *Cat*, the interpreter searches for a sequence of rules to be used using the

[5] From now on, we distinguish control predicates from classical predicates.

following strategy: if the rule category is *pre_rSat* or *post_rSat* then the sequence consists of only the first applicable rule – if there exists, or is empty – otherwise; if the rule category is *rSat* or *rNF* then different sequences of rules will be tried, where short sequences have higher priorities. Having a sequence of rules, the interpreter applies it to *Goal* as described in the previous paragraph. The task is done by the predicate

solve_using_mrules(Cal, Cat, Goal, AtomOut, PostComputation),

where *AtomOut* and *PostComputation* are outputs.

Resolving a modal atom *Goal* with a modal clause in *Calculus* is done according to the framework of SLD-resolution for MProlog, using the required predicates of *Calculus* in an appropriate way. The task is done by the predicate

solve_using_mclauses(Calculus, Goal).

Now return to the problem of resolving *mcall_ (Goal, Calculus)* for the case when *Goal* is a modal atom. It is done by executing the following statements

solve_using_mrules(Calculus, pre_rSat, Goal, A2, F2),
solve_using_mrules(Calculus, rSat, A2, A3, F3),
solve_using_mrules(Calculus, post_rSat, A3, A4, F4),
solve_using_mrules(Calculus, rNF, A4, A5, F5),
solve_using_mclauses(Calculus, A5),
F5, F4, F3, F2.

It remains to discuss the interpretation of the control predicates. In the current version of MProlog, we just adopt the following solution, which does not have a logical basis:

```
mcall_(M:(F1,F2), Cal) :- !, mcall_(M:F1, Cal), mcall_(M:F2, Cal).
mcall_(M:(F1;F2), Cal) :- !, mcall_(M:F1, Cal); mcall_(M:F2, Cal).
mcall_(M:(\+ F), Cal) :- !,
     make_dual_modality(Cal, M, M2), \+ mcall_(M2:F, Cal).
mcall_(M:(F1 -> F2), Cal) :- mcall_(M:F1, Cal)->mcall_(M:F2, Cal).
mcall_(M:if(F1, F2, F3), Cal) :- !,
     call(F1) -> mcall_(M:F2, Cal); mcall_(M:F3, Cal).
```

The interpretation for the case of $M : (F1, F2)$ is sound and complete if M is a modality in labeled form (i.e. M does not contain unlabeled existential modal operators). The interpretation for the case of $M : (F1; F2)$ is also sound.

6 Conclusions

This work presents a new design for modal logic programming. We have designed MProlog to obtain high usefulness, effectiveness, and flexibility. For usefulness: codes, libraries, and most features of Prolog can be used in MProlog programs; for effectiveness: classical fragments are interpreted by Prolog itself, and a number of options can be used for MProlog to restrict the search space; for flexibility: there are three kinds of predicates (classical, modal, *dum*) and we can use and mix different calculi in an MProlog program.

MProlog has a very different theoretical foundation than the existing Molog system. In MProlog, a labeling technique is used for existential modal operators instead of skolemization. We also provide and use new technicalities like normal forms of modalities or pre-orders between modal operators. MProlog also eliminates drawbacks of Molog (e.g., MProlog gives computed answers).

We have implemented SLD-resolution calculi for a number of useful modal logics [11], including all of the multimodal logics of belief considered in this work. The multimodal logics of belief $KDI4_s$, $KDI4_s5$, $KDI45$, $KD4_s5_s$, $KD4I_g5_a$ were first introduced and studied by us for modal logic programming [10]. Some of the implemented SLD-resolution calculi, e.g. the ones for KD, $KD45$, $S5$, $KDI4_s5$, $KD4_s5_s$, $KD45_{(m)}$, are very efficient[6].

Our system is a tool for experimenting with applications of modal logic programming to AI. See [11] for an interesting formulation of the wise men puzzle in MProlog. Our system is also a tool for developing and experimenting with new SLD-resolution calculi for modal logic programming.

Acknowledgements: I would like to thank professors Andreas Herzig and Luis Fariñas del Cerro for a discussion on this paper. I would like also to thank Dr. Rajeev Goré and the anonymous reviewers for many helpful comments and suggestions.

References

1. Akama, S.: A Meta-Logical Foundation of Modal Logic Programming. 1-20-1, Higashi-Yurigaoka, Asao-ku, Kawasaki-shi, 215, Japan, December 1989.
2. Balbiani, P., Fariñas del Cerro, L., Herzig, A.: Declarative Semantics for Modal Logic Programs, *Proceedings of the 1988 International Conference on Fifth Generation Computer Systems*, ICOT, 1988, 507–514.
3. Baldoni, M., Giordano, L., Martelli, A.: A Framework for a Modal Logic Programming, *Joint International Conference and Symposium on Logic Programming*, MIT Press, 1996, 52–66.
4. Fariñas del Cerro, L.: MOLOG: A System that Extends PROLOG with Modal Logic, *New Generation Computing*, **4**, 1986, 35–50.
5. Debart, F., Enjalbert, P., Lescot, M.: Multimodal Logic Programming Using Equational and Order-Sorted Logic, *Theoretical Computer Science*, **105**, 1992, 141–166.
6. Fitting, M., Mendelsohn, R. L.: *First-Order Modal Logic*, Kluwer Academic Publishers, 1999.
7. Garson, J.: Quantification in Modal Logic, in: *Handbook of Philosophical Logic, Volume II* (F. Guenthner, D. Gabbay, Eds.), 1999, 249–307.
8. Lloyd, J.: *Foundations of Logic Programming, 2nd Edition*, Springer-Verlag, 1987.
9. Nguyen, L. A.: A Fixpoint Semantics and an SLD-Resolution Calculus for Modal Logic Programs, *Fundamenta Informaticae*, **55**(1), 2003, 63–100.

[6] For the mentioned logics, the *rSat* operator is either deterministic (for KD, $KD45$, and $KD4_s5_s$) or nondeterministic but with a low branching factor (2 for $KD45_{(m)}$, 3 for $S5$, and m for $KDI4_s5$).

10. Nguyen, L. A.: Multimodal Logic Programming and Its Applications to Modal Deductive Databases, *manuscript (served as a technical report), available on Internet at http://www.mimuw.edu.pl/~nguyen/papers.html*, 2003.
11. Nguyen, L. A.: Source Files, Calculi, and Examples of MProlog, *available on Internet at http://www.mimuw.edu.pl/~nguyen/mprolog*, 2004.
12. Nonnengart, A.: How to Use Modalities and Sorts in Prolog, *Proceedings of JELIA'94, LNCS 838* (C. MacNish, D. Pearce, L. M. Pereira, Eds.), Springer, 1994, 365–378.