Programming in Modal Logic: An Extension of PROLOG based on Modal Logic

Yasubumi SAKAKIBARA
IIAS-SIS, Fujitsu Ltd.
140 Miyamoto, Numazu
Shizuoka, JAPAN 410-03

Abstract

In this paper, we will attempt to give a procedural interpretation to modal logic. Modal logic is used as a programming language and then its procedural interpretation defines a computational procedure for the language. This is done within the framework of logic programming and is one of extensions of PROLOG based on modal logic. Further, we will demonstrate some advantages of the extension such as modurality, hierarchy or structure of logic programs.

1. Introduction

Recently, various forms of logic programming are presented and developed. Logic programming is based on first-order predicate logic and uses it as a programming language. However first-order predicate logic can deal with as its domain only one world in which the truth of an expression does never change and does not depend on place and time. Thus it lacks the expressive power of structure of space, state or time. In order to overcome this shortcoming, various improvements of logic programming are studied [1,2,3,6,8]. In this direction, based on modal logic we will propose an extension of logic programming, and we call it *modal logic programming*. That is to say, we will give a procedural interpretation to modal logic. To do this, we first have to consider semantics of modal logic. Modal logic is one of nonstandard logic in the sense that it is different from first-order logic, and has two kinds of semantics, one of which is axiomatic semantics and the other is possible-world semantics based on the possible-world model developed by Kripke. In our study, we will use the possible-world semantics. Consequently, we will enjoy multi-worlds as modules or as procedures on logic programming. Furthermore, compared with other studies of extensions, we can have the exact logical argument of modal logic programming and never introduce any impure primitive, and this is not the integrated way of two different programming paradigms such as logic programming and object-oriented programming. Thus modal logic

programming is a simple and natural extension of logic programming, and further we will demonstrate that this has a good advantage to analyse structure or modularity of logic programs in the logical framework and to practice program debugging based on logical semantics.

2. Modal logic ([4])

Modal logic is a logic which deals with modality such as necessity or possibility, and its model-theoretic semantics is possible-world semantics based on possible-world model. A possible-world model is a triple (V,R,W) where V is a value assignment, R is an accessibility relation from one world to another and W is a set of worlds. Given two worlds w_i and w_j, w_iRw_j means that w_j is accessible from w_i. The truth of a modal expression is decided based on this relation R. Actually, for a proposition p and a world $w \in W$, p is necessarily true in w (i.e. $V(\Box p,w)=1$) iff for all $w_i \in W$ such that wRw_i, p is true in w_i (i.e. $V(p,w_i)=1$), and p is possibly true in w (i.e. $V(\Diamond p,w)=1$) iff there exists at least one world w_i such that wRw_i and p is true in w_i. Then we can define the validity of a modal expression A : A is valid iff for all possible-world models (V,R,W), $V(A,w)=1$ for all $w \in W$. The correspondence to the proof theoretic semantics is that A is valid iff $S \vdash A$ (A is derived from S) where S is the axiom set for modal logic and the symbol $\vdash$ means a derivation in that axiom system. The differences among some of the most important axiom systems for modal logic correspond exactly to certain restrictions on the accessibility relation of the possible-world models of those systems. This result was proved by Kripke. For example, by restrictions on the accessibility relation to reflexive, symmetric or transitive or these combinations we get several modal axiom systems of type T, S4 or S5.

Now we discuss what the modal logic programming is. There are at least two ways of it. One way is to regard a written program P as axioms and derive theorems T as answers from those axioms. This is analogous to logic programming like PROLOG. On the other hand, it is possible to describe one possible-world model as a program and use modal expressions interpreted in that possible-world model. In this paper, we take the latter and in the following sections we describe it in detail.

3. Programming in modal logic

3.1. Syntax of modal logic program

Definition atom

An *atom* is $p(t_1,...,t_n)$ where p is an n-ary predicate symbol and $t_1,...,t_n$ are terms.

Definition modal atom

A *modal atom* is either $\Box p(t_1,...,t_n)$ or $\Diamond p(t_1,...,t_n)$ where p is an n-ary predicate symbol and $t_1,...,t_n$ are terms.

Definition program clause

A *program clause* is of the form : $A \leftarrow A_1,...,A_n$ where A is an atom and each A_i is an atom or a modal atom ($1 \leqq i \leqq n$).

Modal atoms appear only in the body of a program clause.

Definition program

A *program* is a finite set of program clauses.

Definition world

A *world* definition defines its world name and program in it. A world is of the form :

world <world name> of

<program>

fo.

A program defined in a world is considered as an axiom set which represents various attributes and relations hold in that world. A world is also considered as a module. Thus a world definition provides a framework to express a structure of a program.

Definition relation

A *relation* definition defines an accessibility relation between two worlds. We use a special predicate symbol "re", a relation atom and a relation clause which constitute from them to write an accessibility relation. A *relation atom* is $re(w_1,w_2)$ where each of w_1 and w_2 is a world name or a variable. The meaning of a relation atom defined for two worlds w_i and w_j is $re(w_i,w_j)$ which means that w_j is accessible from w_i. A relation clause is $R \leftarrow R_1,...,R_n$ where $R,R_1,...,R_n$ are relation atoms. A relation is a finite set of relation clauses of the form :

relation of

{a finite set of relation clauses} fo.

We can flexibly define various accessibility relations or structures between worlds by using these relation clauses. This is one of advantages of our way.

Example

relation of

re(w1,w2);

re(w2,w3);

re(X,X); (reflexive)

re(X,Y)←re(Y,X); (symmetric)

re(X,Y)←re(X,Z), re(Z,Y); (transitive)

fo.

In this relation definition, re(w1,w2), re(w2,w3), re(w1,w1), re(w2,w2), re(w3,w3), re(w2,w1), re(w3,w2), re(w1,w3) and re(w3,w1) are true.

Definition modal logic program

A *modal logic program* consists of several world definitions and one relation definition between them, and is of the form :

relation of

{a finite set of relation clauses}

fo.

world <world name> of

<program>

fo.

. . .

world <world name> of

<program>

fo.

3.2. Semantics of modal logic program

In this section, we will give a procedural interpretation based on possible-world semantics for modal logic program. As we mentioned before, we use modal expressions interpreted in the possible-world model expressed by a modal logic program for an extension. First we will give a model-theoretic semantics.

3.2.1. Model-theoretic semantics

Definition Let P be a modal logic program. Let (V,R,W_p) be a possible-world model where W_p is a set of all world names in P.

(1) For w and $w' \in W_p$, a relation atom re(w,w') is *true* in R iff $(w,w') \in R$.

(2) A ground relation clause $R_0 \leftarrow R_1,...,R_n$ is *true* in R iff R_0 is true in R or at least one of $R_1,...,R_n$ is not true in R.

(3) A relation clause is *true* in R iff each of its ground instance is true in R. (A ground instance of a relation clause is obtained by replacing every occurrence of a variable in the relation clause by a world name in W_p.)

(4) A set of relation clauses is *true* in R iff each clause in it is true in R.

We say R is an *accessibility relation model for* the relation in P if the set of relation clauses defined in it is true in R and we write **R**(P). Clearly the intersection of all accessibility relation models for the relation in P exists and we write $\cap$**R**(P). We mean the model-theoretic semantics of the relation in P by $\cap$**R**(P).

Definition Let P be a modal logic program. Let $(V,\cap\mathbf{R}(P),W_p)$ be a possible-world model where W_p is a set of all world names in P.

(1) A ground atom A is true in a world $w \in W_p$ in V iff $V(A,w)=1$.

(2) A ground modal atom $\Diamond A$ is true in w in V iff there exists at least one world w' such that $(w,w') \in \cap\mathbf{R}(P)$ and A is true in w' in V.

(3) A ground modal atom $\Box A$ is true in w in V iff for all $w' \in W_p$ such that $(w,w') \in \cap\mathbf{R}(P)$, A is true in w' in V.

(4) A ground program clause $A_0 \leftarrow A_1,...,A_n$ is true in w in V iff A_0 is true in w in V or at least one of $A_1,...,A_n$ is not true in w in V.

(5) A program clause is true in w in V iff each of its ground instance is true in w in V.

(6) A set of program clauses is true in w in V iff each program clause in it is true in w in V.

There exists the problem of quantifications and identifiers among different possible worlds. We adopt Herbrand interpretations in which constants and functions are literally interpreted and hence each term has the same denotation in every possible world. The domain in every possible world is the Herbrand universe which is the set of all ground terms which can be formed out of the constants and functions appeared in a modal logic program P. Then a ground instance of an atom, a modal atom or a program clause is obtained by replacing every occurrence of a variable in it by a term in the Herbrand universe of P.

We say $(V,\cap\mathbf{R}(P),W_p)$ is a *possible-world model for* P if the set of program clauses defined in each world $w\in W_p$ is true in w in V. For an atom or a modal atom A, we say A is a *logical consequence in* w of P if, for all possible-world models $(V,\cap\mathbf{R}(P),W_p)$ for P, A is true in w in V and we write $P,w\models A$. Thus we mean the model-theoretic semantics of a modal logic program P by $P,w\models A$, and therefore $(V_\models,\cap\mathbf{R}(P),W_p)$ is the possible-world model expressed by a modal logic program P where $V_\models$ is a value assignment such that $V_\models(A,W)=1$ iff $P,W\models A$.

3.2.2. Procedural semantics

Now we define a proof procedure for modal logic program and simultaneously give a procedural interpretation by it.

Definition goal

A *goal* is a list of pairs of the form :

$$((\leftarrow A_1,W_1),\ldots,(\leftarrow A_n,W_n))$$

where each A_i is an atom or a modal atom and W_i is a world name $(1\leqq i\leqq n)$.

Before giving the definition of a proof procedure, we need to enumerate all of ordered pairs (w,w') of world names in P such that (w,w') in $\cap\mathbf{R}(P)$ (i.e. w' is accessible from w). Since the set of all world names W_p is finite, there is an algorithm to enumerate them. Therefore we suppose that the list of them is calculated by some algorithm.

Definition (proof procedure)

Let P be a modal logic program, G be a goal of the form : $((\leftarrow A_1,W_1),\ldots,(\leftarrow A_m,W_m))$ $(m>0)$ and $(\leftarrow A_k,W_k)$ be a selected pair. Let L_p be the list of all of ordered pairs (w,w') in $\cap\mathbf{R}(P)$. Then proof procedure *derives* the new goal as follows :

(a) Suppose A_k is an atom, $B_0\leftarrow B_1,\ldots,B_n$ $(n\geqq 0)$ is a program clause defined in the world W_k and θ is a most general unifier of A_k and B_0. Then the derived goal G' is

$$((\leftarrow A_1,W_1),\ldots,(\leftarrow A_{k-1},W_{k-1}),(\leftarrow B_1,W_k),\ldots,(\leftarrow B_n,W_k),(\leftarrow A_{k+1},W_{k+1}),\ldots,(\leftarrow A_m,W_m))\theta$$

(b) Suppose A_k is a modal atom of the form $\Diamond A$ and (W_k,Wa) in L_p (i.e. Wa is an accessible world from W_k). Then the derived goal G' is

$$((\leftarrow A_1,W_1),\ldots,(\leftarrow A_{k-1},W_{k-1}),(\leftarrow A,Wa),(\leftarrow A_{k+1},W_{k+1}),\ldots,(\leftarrow A_m,W_m))$$

(c) Suppose A_k is a modal atom of the form $\Box A$ and $(W_k,Wa_1),\ldots,(W_k,Wa_h)$ are all of ordered pairs in L_p such that the first element of each ordered pair is W_k. Then the derived goal G' is

$$((\leftarrow A_1,W_1),\ldots,(\leftarrow A_{k-1},W_{k-1}),(\leftarrow A,Wa_1),\ldots,(\leftarrow A,Wa_h),(\leftarrow A_{k+1},W_{k+1}),\ldots,(\leftarrow A_m,W_m))$$

In (b) and (c), θ is the identity substitution.

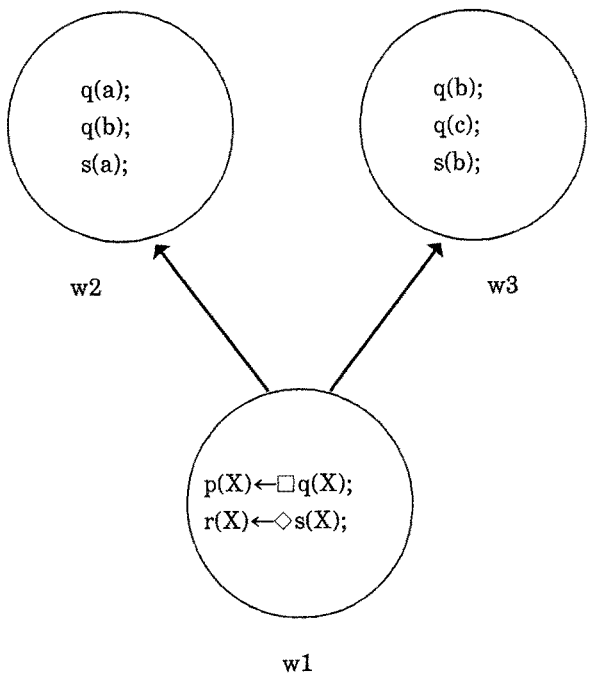
Definition derivation

Let P be a modal logic program and G be a goal. A *derivation* of G is a sequence $G_0 = G$, G_1, G_2, ... of goals such that each G_{i+1} is derived from G_i by proof procedure. A derivation is *successful* if it is finite and its last goal is empty.

For an atom or a modal atom A, we say A is *proved to be true* in a world W if a derivation of $(\leftarrow A, W)$ is successful. By this, we mean the procedural semantics of P.

Example

```
relation of
    re(w1,w2);
    re(w1,w3);
fo.
world w1 of
    p(X)←□q(X);
    r(X)←◇s(X);
fo.
world w2 of
    q(a);
    q(b);
    s(a);
fo.
world w3 of
    q(b);
    q(c);
    s(b);
fo.
```



In the world w1, $\Box q(b)$, $\Diamond q(a)$, $\Diamond q(c)$, $\Diamond s(a)$ and $\Diamond s(b)$ are proved to be true and so are p(b), r(a) and r(b). Thus modal logic programming introduces hierarchic structures into logic programming.

Next our work is to show the soundness and completeness of the above proof procedure. More precisely, it is to show the following : Let P be a modal logic program, A be an atom or a modal atom and W be a world name.

(Soundness) If there exists a successful derivation of (←A,W), then A is a logical consequence in W of P, i.e. P,W⊨A.

(Completeness) If P,W⊨A, then there is a successful derivation of (←A,W).

It is easy to show the soundness. It may take more work to show the completeness.

3.2.3. Semantics of negation

In standard logic program, the negation as failure rule is used to infer negative information. This rule states that if all derivations of ←A are finitely failed, then infer not(A). We define the negation as failure rule in the same way as standard logic program.

Definition finitely failed

Let P be a modal logic program and G be a goal. A derivation of G is *finitely failed* if the derivation is finite and ends with a goal where the atom A of the selected pair (←A,W) does not unify with the head of any program clause defined in the world W.

Then we can state that the negation as failure rule in a modal logic program P is the rule that, for an atom or a modal atom A and a world name W, if all derivations of (←A,W) is finitely failed, then we infer that not(A) is true in W. In this procedural interpretation of "not" and the procedural interpretation of modal operators □ and ◇ defined in the previous section, □A≡not(◇not(A)) can be established.

Now we summarize the special feature of modal logic programming. A modal logic program expresses a possible-world model and modal expressions in it is procedurally interpreted in the possible-world model. The proof procedure may correspond to the "semantic tableau method". And various relationships among worlds can be defined by relation definitions. Furthermore modal logic programming can keep its logical consistency. This is one of the advantages compared with other ways such as an integrated way of logic programming and object-oriented programming or a way of introducing impure primitives like module into logic programming. This is very useful for the program debugging.

4. World as Module or Procedure ?

From the programming language point of view, we can consider worlds as modules or procedures. Then a world limits the scope of validity of each predicate name, and a relation definition limits the scope of availability of the program in each world(i.e. it defines the accessibility relation of that program). These introduce program structures and also

hierarchization into logic programming. Thus we can use multi-worlds mechanism in modal logic programming. These also realize the polysemy of a predicate name.

In the next section, we will describe a simple interpreter for modal logic program.

5. A Simple Interpreter for modal logic program

```
solve(true, W) :- !, true.
solve((A, B), W) :- !, solve(A, W), solve(B, W).
solve(◇A, W) :- !, rel(W, W1), solve(A, W1).
solve(□A, W) :- !, allrel(W, W1), allsolve(A, W1).
solve(A, W) :- !, clause(<A←B, W>), solve(B, W).
allsolve(A, []).
allsolve(A, [W | W1]) :- solve(A, W), allsolve(A, W1).
rel(W, W1) :- solve(re(W, W1), relation).
allrel(W, W1) :- W1 is a list of all worlds w such that re(W, w).
```

where it assumes that the modal logic program P is represented as clauses "clause(<A←B, W>)" for any program clause A←B in the world W and the relation definition is represented as clauses "clause(<A←B, relation>)" for any relation clause A←B.

For example, the above interpreter exactly corresponds to the axioms in Moore's first-order theory of knowledge [5] if we consider a □ operator as a KNOW operator and a compatible relation K as an accessible relation re.

Here we also describe a simple backtrace program debugger which is a modification of the Shapiro's contradiction backtracing algorithm [7].

```
backtrace((A,B),W,CE) :- !, backtrace(A,W,CE), backtrace(B,W,CE).
backtrace(◇A,W,CE) :- !, rel(W,W1), backtrace(A,W1,CE).
backtrace(□A,W,CE) :- !, allrel(W,W1), allbacktrace(A,W1,CE).
backtrace(A,W,CE) :-
    clause(<A←B,W>), backtrace(B,W,CE), resolve((A←B),W,CE).
allbacktrace(A,[],CE).
allbacktrace(A,[W | W1],CE) :- backtrace(A,W,CE), allbacktrace(A,W1,CE).
resolve((A←B),W,CE) :- var(CE), !,
    ask(A,W,V), (V=false, CE=(W,A←B); V=true).
```

```
resolve((A←B),W,CE).
ask(A,W,V) :- recordedfact(A,W,V), !.
ask(A,W,V) :- repeat, write(A), write(in), write(W), put(63), nl,
                read(V), (V=true; V=false), assert(recordedfact(A,W,V)).
```

6. Comparison with other studies

The MOLOG [3] is an extension of PROLOG using modal logic and so very similar to our direction. However its system axiomatically interprets modal operators and so different from our way based on possible-world semantics. They give a proof theory for the universal modal operator know(a) in the system S5. Prolog/KR [6] and metaProlog [1] are very related to and have inspired our study. Prolog/KR has the concept of "worlds" and is also based on modal logic semantics. The metaProlog is also an extension to consider metatheory and has "theories" like our worlds.

7. Concluding Remarks

One of further issues which we are now working on is to extend modal logic programming so as to deal with several modalities simultaneously. Specifically, by adding to modal operators an extra argument for representing a name of a modal logic system (i.e. name of a relation), we will extend modal logic programming so that we may describe several relations each of which corresponds to each modal logic system. Following illustrates some images of the extension :

```
relation m1 of
   . . .
fo.
relation m2 of
   . . .
fo.
   . . .
world w1 of
   p←m1 pos q(X);
   p←m2 nec s(X), m4 pos t(b);
      . . .
fo.
```

. . .

where nec and pos are infix operators which respectively represent necessity and possibility and have two arguments. For example, "m1 nec" means the necessity operator in m1 modal system.

Acknowledgements

The author would like to thank Dr. T.Kitagawa, the president of IIAS-SIS, Dr. H.Enomoto, the director of IIAS-SIS, for giving us the opportunity to pursue this work and helping us with it. He is deeply grateful to Dr. T.Yokomori, IIAS-SIS, for reading the draft of this paper and giving us many valuable comments. He is also indebted to Mr. Y.Takada, IIAS-SIS, for many fruitful discussions and comments.

This work is part of the major R&D of the FGCP, conducted under program set up by MITI.

References

[1] Bowen, K.A. : Meta-Level Programming and Knowledge Representation, *New Generation Computing*, 3(1985), 359-383.

[2] Chikayama, T. : ESP Reference Manual, *ICOT Technical Report TR-044*, 1984.

[3] Fariñas del Cerro, L : MOLOG : A System That Extends PROLOG with Modal Logic, *New Generation Computing*, 4(1986), 35-50.

[4] Hughes, G. E., & Cresswell, M. J. : An Introduction to Modal Logic, London : Methuen.

[5] Moore, R. C. : A Formal Theory of Knowledge and Action, in *Formal Theories of the Commonsense World* (J.R.Hobbs, & R.C.Moore, Eds.), Ablex, New Jersey, 319-358.

[6] Nakashima, H. : Prolog/KR - language features, *Proc. of the 1st Intern. Logic Programming Conference*, Marseille, 1982, 65-70.

[7] Shapiro, E. Y. : Inductive Inference of Theories from Facts, *Research Report 192*, Dept. of Computer Science, Yale University, 1981.

[8] Warren, D. S. : Database Updates in Pure PROLOG, *Proc. of the Intern. Conf. on FGCS' 84*, Tokyo, 1984, 244-253.