A Modal Extension of Logic Programming

Matteo Baldoni, Laura Giordano and Alberto Martelli
Dipartimento di Informatica - Università di Torino
C.so Svizzera 185 - 10149 TORINO
E-mail: {baldoni,laura,mrt}@di.unito.it

Abstract

In this paper we present a modal extension of logic programming, which provides reasoning capabilities in a multiagent situation. The language contains embedded implications, modal operators $[a_i]$ to represent agent beliefs, together with a kind of "common knowledge" operator. In this language we can also define modules, compose them in several ways, and, also, we can perform hypothetical reasoning.

1 Introduction

The problem of extending logic programming languages with modal operators, has been studied by several researchers. In particular, in [4] an extension of Prolog with modal operators, called MOLOG, is proposed, and a resolution procedure, close to Prolog resolution, is defined for modal Horn clauses in the logic S5 which contain universal modal operators of the form Know(a). Another modal extension of logic programming is the temporal logic programming language TEMPLOG introduced in [1]. Moreover, in [20] a logic language extended with a modal operator *assume* is defined, which allows atomic updates to be performed on the program.

Recently we have studied how structuring facilities, like blocks and modules, can be introduced in logic programming languages by making use of modal extensions [9, 2]. In particular, in [9] we have focused on a language with *embedded implications*, like the languages in [7, 6, 14, 12, 8]. These languages allow implications of the form $D \rightarrow G$ to occur both in goals and in clause bodies, and this provides a way of introducing local definitions of clauses: the clauses in D are intended to be local to the goal G, as they can be used only in a proof of G. The meaning of hypothetical implications, is that of hypothetical insertion: the goal $D \rightarrow G$ is derivable from a program P if G is derivable from a new program updated with D. When intuitionistic logic is taken as the underlying logic of this language, like in N_Prolog and in λ-Prolog , embedded implications allow hypothetical reasoning to be performed.

In [8] we have shown how different languages with embedded implications can be obtained by choosing different visibility rules for locally defined clauses. A modal extension of Horn clause logic (based on S4 logic) can provide a unifying framework in

which these different kinds of local definitions of clauses can be defined and integrated [9]. This extension provide a notion of block, from which various kinds of modules can also be defined, by introducing some syntactic sugar.

In [2] we have proposed a modal extension of Horn clause logic which provides modules as a basic feature. This extension is defined on the line of the above mentioned language with blocks, though in this case the language does not contain embedded implications. On the contrary, modules are defined by introducing different modal operators ($[m_1], \ldots, [m_k]$) each one associated with a module. Module composition can be obtained by allowing modules to export clauses or derived facts. To achieve this purpose, a different modal operator $\Box$ is introduced, which makes possible to distinguish clauses local to a module from those that are fully exported and those whose consequences are exported. This language allows one to model different kinds of modules presented in the literature (so that in each situation the kinds of module that suit better can be adopted); furthermore, this language provides some well known features of object oriented programming, like the possibility of representing dependencies among modules in a hierarchy, and the notion of *self* to reason on this hierarchy.

Although these extensions were introduced with the aim of defining modules, the modal operators introduced in this language can be given alternative meanings, which can allow a more general use of the language. For instance, the language can be suitable for representing knowledge and belief by considering the modal operators $[m_i]$ as "belief" operators, and the $\Box$ operator as a kind of "common knowledge" operator. The aim of this paper is to integrate all previously mentioned extensions (modal operators and hypothetical implications) in an extended language to be used for knowledge representation. The language should be as general as possible, while retaining the distictive feature of logic programming of having a goal directed operational semantics.

The logic programming language presented in this abstract is a modal logic refinement of hereditary Harrop formulas [17], and it lies on the same line as other logic programming languages which are not based on classical first-order logic, like those based on intuitionistic logic [7, 6, 12, 13, 14, 16, 15] higher-order logic [17] and linear logic [11]. This language we define subsumes the languages defined in [9, 2], so that it allows both to define module constructs and to perform hypothetical reasoning. Moreover it provides a simple way to formulate reasoning capabilities in a multiple agent situation.

2 The Language

In this section we introduce a modal logic programming language which subsumes the languages defined in [9, 2], so that it allows both to define module constructs and to perform hypothetical reasoning. Moreover it provides a simple way to formulate reasoning capabilities in a multiple agent situation. We extend Horn clause language, with k modal operators $[a_1], \ldots, [a_k]$, where the a_i's are constants, each one representing an agent, and a modal operator $\Box$, which is a sort of common knowledge operator. Each modal formula $[a_i]\alpha$ can be read "agent i believes α".

Let A be an atomic formula and T a distinguished proposition (true). The syntax of the language is the following:

$$\begin{aligned} G ::= \; & T \mid A \mid G_1 \wedge G_2 \mid \exists x G \mid [a_i]G \mid \Box G \mid [a_i](D \supset G) \mid \\ & \Box(D \supset G) \\ D ::= \; & G \supset H \mid [a_i]D \mid \Box D \mid \forall x D \\ H ::= \; & A \mid [a_i]H \mid \Box H \ , \end{aligned}$$

where G stands for a goal, D for a clause, and H for a clause head. In the following D will be interchangeably regarded as a conjunction and a set of clauses. A program P consists of a set of clauses D.

In this language, modal operators $[a_i]$ and $\Box$ can freely occur in front of clauses, in front of clause heads, in front of each goal and, in particular, in front of embedded implications (or implication goals). $[a_i]G$ is a goal and means that G has to be proved in the beliefs of agent i while $[a_i]D$ means that the clause D is part of the beliefs of agent i. As regards implication goals, they must be preceded by at least one modal operator. Note that the language is extended with respect to the one presented in [2], where the definition of clauses were restricted to few different shapes (in particular, arbitrary sequences of modal operators were not admitted) and embedded implications were not allowed.

As an example consider the following formulation of the *(two) wise men puzzle*:

(1) $\Box \ (bs(b) \supset ws(a))$

(2) $\Box(bs(b) \supset [a]bs(b))$

(3) $\Box([b]((T \supset bs(b)) \supset \bot) \supset [b]ws(b))$

(4) $[b]([a]ws(a) \supset \bot)$

In this example, $\bot$ is a distinguished proposition representing falsity; $bs(a)$ ($ws(a)$) represents the fact that a has a black (white) spot on his forehead. [a] is a modal operator and $[a]F$ means that agent a knows F. All the clauses preceded by the modal operator $\Box$, denote the information which is common to all agents. In clause (3), a form of negation has been introduced as usually in a language with embedded implications, by making use of the proposition $\bot$: *"not F"* is expressed by the implication $F \supset \bot$. So clause (3) can be read: if agent b knows that he has not the black spot, then he knows that he has the white spot. (4) says that b knows that a doesn't know if he has the white spot on his forehead. This models the situation in which a is the first which is asked, he doesn't know the answer and b knows that. In this example, while the modal operators $[a]$, $[b]$ and $\Box$ give a way to distinguish among information of the single agents and information common to all of them, embedded implications allow forms of hypothetical reasoning to be performed.

Following [2], in our language the modal operators $[a_i]$ are all ruled by the axioms of K, and, in particular,

$$[a_i](B \supset A) \supset ([a_i]B \supset [a_i]A)$$

is part of the axiomatization, for each $[a_i]$. On the other hand, the operator $\Box$ is ruled by the axioms of S4. Thus the axiomatization contains the following axioms:

(K) $\Box(B \supset A) \supset (\Box B \supset \Box A)$
(T) $\Box A \supset A$
(4) $\Box A \supset \Box\Box A$.

Moreover, since the operator $\Box$ is intended to represents what is known by all the agents, it interacts with each operator $[a_i]$ by the following interaction axioms,

$\Box A \supset [a_i]\Box A$.

This language is quite similar to the modal language introduced in [10] for dealing with the notions of knowledge and common knowledge, though, the modal operator $\Box$ we have introduced does not exactly coincide with (and it is weaker than) the common knowledge operator in [10]. In particular, it holds that, for all modal operators $[a_i]$, $\Box\alpha \supset [a_i]\Box\alpha$, so that if $\Box\alpha$ holds, then all the agents believe $\Box\alpha$. However, it does not hold that $[a_1]\Box A \wedge \ldots \wedge [a_k]\Box A \wedge A \supset \Box A$ while it is expected to hold when $\Box$ is the common knowledge operator. In [2] we have defined both Kripke semantics and a sequent calculus for a modal language L containing the logical conectives $\neg$, $\wedge$, $\supset$, $\forall$ and $\exists$, and the modal operators $[a_1], \ldots, [a_k]$ and $\Box$. In the next section, we will recall the sequent calculus.

3 Sequent calculus

For simplicity it has been assumed that the language L does not contain function symbols, but only constants. Let's assume that the language L contains countably many constants, variables and relational symbols. Since, as mentioned above, the modal operator $\Box$ is of type S4, we present a cut-free sequent calculus for the language L which extends the cut-free sequent calculus for S4 presented in [19] (section 2.1) and adapted from [5]. In addition to the rules of the calculus for S4, a new rule is needed to deal with each modal operator $[a_i]$.

As a difference with the calculus in [19], and also with the calculus we have proposed in [2], we will not include the rules for negation $(L\neg)$ and $(R\neg)$, since this not the kind of negation we want in our language. Here, we only consider the positive fragment of the language in [2]. As mentioned above, we introduce a form of negation in the language, by making use of implication and of the distingiushed symbol $\bot$, representing falsity. As in minimal logic, $\bot$ is not required to satisfy any particular property. The calculus is the following:

$$\frac{}{\Gamma, A \to A, \Delta}$$

$$\frac{\Gamma, A, B \to \Delta}{\Gamma, A \wedge B \to \Delta}(L\wedge) \qquad \frac{\Gamma \to B, \Delta \quad \Gamma \to C, \Delta}{\Gamma \to B \wedge C, \Delta}(R\wedge)$$

$$\frac{\Gamma \to A, \Delta \quad \Gamma, B \to \Delta}{\Gamma, A \supset B \to \Delta}(L\supset) \qquad \frac{\Gamma, A \to B, \Delta}{\Gamma \to A \supset B, \Delta}(R\supset)$$

$$\frac{\Gamma, [x/c]A \to \Delta}{\Gamma, \forall x A \to \Delta}(L\forall) \qquad \frac{\Gamma \to [x/a]A, \Delta}{\Gamma \to \forall x A, \Delta}(R\forall)$$

$$\frac{\Gamma, [x/a]A \to \Delta}{\Gamma, \exists x A \to \Delta}(L\exists) \qquad \frac{\Gamma \to [x/c]A, \Delta}{\Gamma \to \exists x A, \Delta}(R\exists)$$

$$\frac{\Gamma, A \to \Delta}{\Gamma, \Box A \to \Delta}(L\Box) \qquad \frac{\Gamma^* \to A}{\Gamma \to \Box A, \Delta}(R\Box)$$

$$\frac{\Gamma^*, \Gamma_i^* \to A}{\Gamma \to [a_i]A, \Delta}(R[a_i])$$

where $\Gamma^* = \{\Box\alpha : \Box\alpha \in \Gamma\}$ and $\Gamma_i^* = \{\alpha : [a_i]\alpha \in \Gamma\}$. For $(R\forall)$ and $(L\exists)$ here is the proviso that a is a parameter that does not occur in any formula of the lower sequent. In rule $(L\forall)$ and $(R\exists)$ c is any constant of the language.

In this sequent calculus there is no need for structural rules, since in a sequent $\Gamma \to \Delta$ the antecedent and the succedent are sets of statements rather than sequences of statements.

Since T is a distinguished symbol which can be regarded as any propositional tautology, we can assume to have the additional initial sequent $\Gamma \to T, \Delta$ to deal with this symbol.

A proof for the sequent $\Gamma \to \Delta$ is a finite tree constructed using the above rules, having the root labelled with $\Gamma \to \Delta$ and the leaves labelled with initial sequents, i.e. sequents of the form $\Gamma, A \to A, \Delta$ or of the form $\Gamma \to T, \Delta$.

Note that the rules $(L\Box)$ and $(R\Box)$ above are exactly the same as those of the calculus for S4 in [19]. The only difference is due to the fact that in our language L the existential modal operator $\Diamond$ is not present. The inference rule $(L\Box)$ is needed since the modal operator $\Box$ is of type $S4$, and, in the Kripke semantics, the accessibility relation associated with it is reflexive.

The inference rule $(R[a_i])$ is needed to deal with the modal operator $[a_i]$, and it is quite similar in style to the rule $(R\Box)$. As a difference with $(R\Box)$, the antecedent of its premise contains both the set Γ^* and the set Γ_i^* . This corresponds to the fact that all the formulas of type $\Box\alpha$ are visible from within the context of a modal operator $[a_i]$. Moreover, since the modal operators $[a_i]$ are of type K, and, in the Kripke semantics, the accessibility relation associated with each of them is not reflexive, there is no rule $(L[a_i])$.

4 A goal directed proof procedure

A goal directed proof procedure can be defined for this language with modal operators. Following [16], in order to avoid problems with variable renaming and substitutions, given a set of clauses D, we denote by $[D]$ the set of all ground instances of the clauses in D. We introduce a notion of operational derivability of a closed goal G from a program P by induction on the structure of G. Since the language allows

modal operators, in the operational semantics we will introduce a notion of operational derivability of a closed goal G from a program P in a certain modal context. A modal context is a sequence of modal operators $L_1|...|L_n$, which represents the sequence of right rules (($R\Box$) or ($R[a_i]$)) that have been applied in the corresponding sequent proof, up to that point. To deal with embedded implications, each modal operator L_j in the sequence is associated with a (possibly empty) set of clauses D_j.

Hence, we define the operational derivability of a closed goal G from a list of pairs

$$(L_0, D_0) \mid (L_1, D_1) \mid \ldots, (L_n, D_n)$$

(where, for uniformity, D_0 is the initial program P, and L_0 is not used), by induction on the structure of G, as follows (we will denote by Γ an arbitrary sequence of modal operators):

1. $\langle (L_0, D_0) \mid (L_1, D_1) \mid \ldots \mid (L_n, D_n) \rangle \vdash T$;

2. $\langle (L_0, D_0) \mid (L_1, D_1) \mid \ldots \mid (L_n, D_n) \rangle \vdash A$ if for some $j = 0, \ldots, n$, there is a clause $\Gamma_b(G \supset \Gamma_h A) \in [D_j]$ and a k, $j \leq k \leq n$, such that
   Γ_b matches $\Gamma_1 = L_{j+1} \ldots L_k$,
   Γ_h matches $\Gamma_2 = L_{k+1} \ldots L_n$, and
   $\langle (L_0, D_0) \mid (L_1, D_1) \mid \ldots \mid (L_k, D_k) \rangle \vdash G$;

3. $\langle (L_0, D_0) \mid (L_1, D_1) \mid \ldots \mid (L_n, D_n) \rangle \vdash G_1 \wedge G_2$ if
   $\langle (L_0, D_0) \mid (L_1, D_1) \mid \ldots \mid (L_n, D_n) \rangle \vdash G_1$
   and $\langle (L_0, D_0) \mid (L_1, D_1) \mid \ldots \mid (L_n, D_n) \rangle \vdash G_2$;

4. $\langle (L_0, D_0) \mid (L_1, D_1) \mid \ldots \mid (L_n, D_n) \rangle \vdash L_{n+1} G$ if
   $\langle (L_0, D_0) \mid (L_1, D_1) \mid \ldots \mid (L_n, D_n) \mid (L_{n+1}, \emptyset) \rangle \vdash G$;

5. $\langle (L_0, D_0) \mid (L_1, D_1) \mid \ldots \mid (L_n, D_n) \rangle \vdash L_{n+1}(D \supset G)$ if
   $\langle (L_0, D_0) \mid (L_1, D_1) \mid \ldots \mid (L_n, D_n) \mid (L_{n+1}, D) \rangle \vdash G$;

6. $\langle (L_0, D_0) \mid (L_1, D_1) \mid \ldots \mid (L_n, D_n) \rangle \vdash \exists x G$ if, for some closed term t,
   $\langle (L_0, D_0) \mid (L_1, D_1) \mid \ldots \mid (L_n, D_n) \rangle \vdash [x/t]G$.

In the definition above, the rules for the conjunction and existentially quantified goals are the usual ones. A goal $[a_i]G$ is proved by adding the modal operator $[a_i]$ to the current context and proving G in the resulting context. The same holds for the modal operator $\Box$.

Embedded implications are always preceded by a modal operator. To prove a goal $L_{n+1}(D \supset G)$, where L_{n+1} is an arbitrary modal operator, the modal operator L_{n+1} is added to the current context together with the clauses in D, and G is proved in the resulting context.

To prove an atomic goal A a clause is selected in the initial program D_0 or from a set of clauses D_j that has been added in the context, by proving some implication goal. In both cases, to verify that the clause is applicable in the current context, it must be checked whether the modal operators in the clause (both in front of it and in front of its head) match the current context, from L_{j+1} to L_n. In particular, the sequence of the modal operators Γ_b in front of the selected clause must match a prefix of the sequence $L_{j+1} \dots L_n$, while the sequence of the modal operators Γ_h in front of the head of the selected clause must match the remaining part of the sequence. We say that a sequence Γ of modal operators matches another sequence Γ' if each modal operator in Γ matches a sequence of operators in Γ' in the ordering. More precisely, each modal operator $[a_i]$ in Γ matches the operator $[a_i]$ itself, while each operator $\Box$ in Γ may match an arbitrary (possibly empty) sequence of modal operators in Γ'.

Formally, a sequence $\Gamma = L_1 \dots L_n$ of modal operators matches another sequence Γ' if each modal operator L_j in Γ can be associated with a sequence of modal operators $f_j(L_j)$ such that the following condition hold:

- $f_j([a_i]) = [a_i]$, for all modal operators $[a_i]$;
- $f_j(\Box)$ is any sequence (including the empty sequence of modal operators);
- $f_1(L_1) \dots f_n(L_n) = \Gamma'$.

Given a program P and a goal G, we say that G is provable from P, if

$$\langle (L_0, P) \rangle \vdash G$$

can be derived by applying the operational rules above. It is possible to prove that the above operational semantics is sound and complete with respect to the sequent calculus above, i.e., $\langle (L_0, P) \rangle \vdash G$ iff the sequent $P \rightarrow G$ is provable.

5 Examples

Example 1 We give a formulation of the *(three) wise men puzzle*. The formulation is quite similar to the one of the two wise man puzzle in section 2. However, in order to avoid introducing many variant of the same clause for the different agents, we make use of parametric modal operators as a shorthand.

(1) $\Box\ \forall X, Y, Z(bs(X) \wedge bs(Y) \wedge X \neq Y \wedge X \neq Z \wedge Y \neq Z \supset ws(Z))$

(2) $\Box \forall X, W(bs(X) \wedge X \neq W \supset [W]bs(X))$

(3) $\Box \forall X([X]((T \supset bs(X)) \supset \bot) \supset [X]ws(X))$

(4) $\Box([a]ws(a) \supset \bot)$

(5) $[c]([b]ws(b) \supset \bot)$

We have modelled the situation in which a, b and c are asked in this ordering. a doesn't know if he has the white spot on his forehead; b doesn't know too; c knows about their answers (clauses (4) and (5)) and, hence, he can conclude that he has the white spot himself.

The goal $G = [c]ws(c)$ succeeds from the program, with the following derivation:

$\langle (L_0, P) \rangle \vdash [c] ws(c)$

$\langle (L_0, P) \mid ([c], \emptyset) \rangle \vdash ws(c)$

$\langle (L_0, P) \rangle \vdash [c]((T \supset bs(c)) \supset \bot)$, by clause (3) with $X = c$

$\langle (L_0, P) \mid ([c], T \supset bs(c)) \rangle \vdash \bot$

$\langle (L_0, P) \mid ([c], T \supset bs(c)) \rangle \vdash [b] ws(b)$, by clause (5)

$\langle (L_0, P) \mid ([c], T \supset bs(c)) \mid ([b], \emptyset) \rangle \vdash ws(b)$

$\langle (L_0, P) \mid ([c], T \supset bs(c)) \rangle \vdash [b]((T \supset bs(b)) \supset \bot)$,
by clause (3) with $X = b$

$\langle (L_0, P) \mid ([c], T \supset bs(c)) \mid ([b], T \supset bs(b)) \rangle \vdash \bot$

$\langle (L_0, P) \mid ([c], T \supset bs(c)) \mid ([b], T \supset bs(b)) \rangle \vdash [a] ws(a)$, by clause (4)

$\langle (L_0, P) \mid ([c], T \supset bs(c)) \mid ([b], T \supset bs(b)) \mid ([a], \emptyset) \rangle \vdash ws(a)$

$\langle (L_0, P) \mid ([c], T \supset bs(c)) \mid ([b], T \supset bs(b)) \mid ([a], \emptyset) \rangle \vdash$
$bs(X) \wedge bs(Y) \wedge X \neq Y \wedge X \neq a \wedge Y \neq a$ by clause (1)

Let us consider separately the subqueries $bs(X)$. We have the following subderivations:

$\langle (L_0, P) \mid ([c], T \supset bs(c)) \mid ([b], T \supset bs(b)) \mid ([a], \emptyset) \rangle \vdash bs(X)$

$\langle (L_0, P) \mid ([c], T \supset bs(c)) \mid ([b], T \supset bs(b)) \rangle \vdash bs(X)$,
by clause (2) with $W = a$

$\langle (L_0, P) \mid ([c], T \supset bs(c)) \mid ([b], T \supset bs(b)) \rangle \vdash T$, by taking $X = b$

and similarly:

$\langle (L_0, P) \mid ([c], T \supset bs(c)) \mid ([b], T \supset bs(b)) \mid ([a], \emptyset) \rangle \vdash bs(Y)$

$\langle (L_0, P) \mid ([c], T \supset bs(c)) \mid ([b], T \supset bs(b)) \rangle \vdash bs(Y)$,
by clause (2) with $W = a$

$\langle (L_0, P) \mid ([c], T \supset bs(c)) \rangle \vdash bs(Y)$, by clause (2) with $W = b$

$\langle (L_0, P) \mid ([c], T \supset bs(c)) \rangle \vdash T$, by taking $Y = c$

Hence, $bs(X) \wedge bs(Y)$ succeeds with $X = b$ and $Y = c$. Thus, $X \neq Y \wedge X \neq a \wedge Y \neq a$ succeeds too. And this concludes the proof.

Example 2 As mentioned above, the modal language we have defined in this paper subsumes the language proposed in [2] to introduce module constructs in Horn clause logic. In particular, the language in [2] associates a modal operator $[m_i]$ with each module and it provides module definitions of the form $\Box[m_i]D$ (where D is a set of clauses), with the meaning that the clauses in D belong to module $[m_i]$. The operator $\Box$ in front of the module definition make it visible from inside other modules. The modal operators $[m_i]$ may occur in front of goals: $[m_i]G$ is a goal and it means

that G has to be proved in the module $[m_i]$. This provides a simple way to define a flat collection of modules and to specify the proof of a goal in a module. In a simplistic view, modules are closed environments. However, different forms of module composition can be obtained by making use of the modal operator $\Box$ to distinguish among clauses local to a module and clauses exported by a module.

Consider for instance the query $[m_1][m_2][m_3]G$. the goal G must be proved in the composition of modules $[m_1]$, $[m_2]$ and $[m_3]$. When modules are regarded as being open, each module may export information to the modules following it in the sequence. The language in [2] makes a distintion among clauses that are *local* to the module in which they are defined, $G \supset A$, clauses that are wholly exported by the module, $\Box(G \supset A)$, and clauses which only export their head, $G \supset \Box A$. Notice that the syntax we have defined in section 2, gives much more freedom, since an arbitrary sequence of modal operators may occur in front of a clause and its head.

Let us consider the following example (taken from [3]), describing inheritance in a hierarchy of modules:

$\Box[animal]$

$\{T \supset \Box mode(walk)$

$\Box(no\_of\_legs(2) \supset mode(run))$

$\Box(no\_of\_legs(4) \supset mode(gallop))\ \}$

$\Box[bird]$

$\{T \supset \Box no\_of\_legs(2)$

$T \supset \Box covering(feather)\ \}$

$\Box[tweety]$

$\{T \supset owner(fred)\ \}.$

The goal

$$[animal][bird][tweety]mode(run)$$

succeeds, since the clause defining $mode(run)$ is exported by the module $[animal]$ and its body can be evaluated in the current context, inculding module $[bird]$ which contains the information $no\_of\_legs(2)$. The goal would have failed, if the alternative clause $no\_of\_legs(2) \supset mode(run)$ had been contained in module $[animal]$. By using clauses preceded by the operator $\Box$ we can achieve a result quite similar to the use of *self* in object-oriented languages.

The more extended language we have defined in this paper, offers some additional features in defining modules. The possibility of using embedded implications in the language can be usefully exploited to specify module interfaces. Indeed, embedded implications can be used to hide predicates of a module that we do not want to export. For instance, let m be a module containing a set of clause definitions

$G_1 \supset p_1$

$\ldots$

$G_n \supset p_n$

together with a set of clauses $D = \{D_1, \ldots, D_k\}$ that must be visible only from within the module m. We can define m as follows:

$$\begin{array}{l} \Box[m] \\ \quad \{(\Box(D \supset G_1) \supset p_1) \\ \quad \ldots \\ \quad (\Box(D \supset G_n) \supset p_n)\}. \end{array}$$

Embedded implications can also be used to couple inheritance and hypothetical reasoning, as done in [3].

Since in this modal language clauses can be preceded by an arbitrary sequence of modal operators, we can generalize module definitions $\Box[m_i]D$ above, by allowing *nested* module definitions as follows:

$$\Box[m_i]\Box[m_j]D,$$

where the module m_j is defined locally to m_i, and it becomes visible whenever m_i is entered.

Example 3 We present the Fibonacci example from [1]. We use a modal operator $[next]$ to represent the next instant of time. We want $fib(x)$ to hold after n instants of time, if x is the n-th Fibonacci number. The formulation is the following:

1. $T \supset fib(0)$
2. $T \supset [next]fib(1)$
3. $\Box(fib(Y) \wedge [next]fib(Z) \wedge X \ is \ Y + Z \supset [next][next]fib(X))$.

Clause (1) says that at time 0, $fib(0)$ holds; clause (2) says that at time 1, $fib(1)$ holds; clause (3) says that, for any time n, if $fib(Y)$ holds at time n, and if $fib(Z)$ holds at time $n+1$, then $fib(X)$, with $X = Y + Z$, holds at time $n+2$.

From this program, the query

$$[next][next][next]fib(X)$$

succeeds with $X = 2$, and indeed 2 is the 3-rd Fibonacci number.

6 Conclusions

In this paper we have defined a modal language which extends the languages proposed in [9, 2] to deal with blocks and modules. This language provides a simple way to formulate reasoning problems in a multiagent situation and it also provides hypothetical reasoning capabilities.

The language we have focused on contains universal modal operators $[a_i]$ of type K, and a modal operator $\Box$ which is a sort of common knowledge operator. Similar but

different languages could be defined by changing the properties of the modal operators $[a_i]$. For instance, the modal operators $[a_i]$ could follow the rules of $K4$ or $S4$, instead of the rules of K. In this cases it could still be possible to define an operational top down semantics similar to the one presented in section 4, by modifying the notion of *matching* between sequences of modal operators. A study of goal directed proof methods in the logic K, K4, S4, and also for S5, has been done in [18] for the fragment of strict implication.

Acknowledgement

This work has been partially supported by CNR - Progetto Finalizzato "Sistemi Informatici e Calcolo Parallelo" under grant n. 92.01577.PF69.

References

[1] M. Abadi and Z. Manna. Temporal logic programming. *J.Symbolic Computation*, (8):277–295, 1989.

[2] M. Baldoni, L.Giordano, and A.Martelli. A multimodal logic to define modules in logic programming. In *Proc. 1993 International Logic Programming Symposium*, pages 473–487, Vancouver, 1993.

[3] Antonio Brogi, Evelina Lamma, and Paola Mello. Inherritance and hypothetical reasoning in logic programming. In *Proc. European Conference on Artificial Intelligence*, pages 105–110, Stockholm, 1990.

[4] L. Fariñas del Cerro. Molog: A system that extends Prolog with modal logic. *New Generation Computing*, (4):35–50, 1986.

[5] M. Fitting. *Proof Methods for Modal and Intuitionistic Logics*, volume 169 of *Synthese library*. D. Reidel, Dordrecht, Holland, 1983.

[6] D. M. Gabbay. NProlog: An extension of Prolog with hypothetical implications.ii. *J.Logic Programming*, 2(4):251–283, 1985.

[7] D. M. Gabbay and N. Reyle. NProlog: An extension of Prolog with hypothetical implications.i. *Journal of Logic Programming*, (4):319–355, 1984.

[8] L. Giordano, A. Martelli, and G.F. Rossi. Extending Horn clause logic with implication goals. *Theoretical Computer Science*, 95:43–74, 1992.

[9] Laura Giordano and Alberto Martelli. A modal reconstruction of blocks and modules in logic programming. In *Proc. 1991 Int. Logic Programming Symposium*, pages 239–253, San Diego, October 1991.

[10] J. Y. Halpern and Y. Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54:319–379, 1992.

[11] J. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic. In G. Kahn, editor, *Sixth Annual Symposium on Logic in Computer Science*, pages 32–42, Amsterdam, 1991.

[12] L. T. Mc Carty. Clausal intuitionistic logic. i. fixed-point semantics. *J. Logic Programming*, 5(1):1–31, 1988.

[13] L. T. Mc Carty. Clausal intuitionistic logic.ii.Tableau proof procedure. *J. Logic Programming*, 5(2):93–132, 1988.

[14] D. Miller. A theory of modules for logic programming. In *Proc. IEEE Symp. on Logic Programming*, pages 106–114, September 1986.

[15] D. Miller. Lexical scoping as universal quantification. In *Proc. 6th Int. Conf. on Logic Programming*, pages 268–283, Lisbon, 1989.

[16] D. Miller. A logical analysis of modules in logic programming. *Journal of Logic Programming*, (6):79–108, 1989.

[17] D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as foundations for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[18] N. Olivetti and D. Gabbay. Goal-directed method for strict implication. Technical report, 1993.

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

[20] D. S. Warren. Database updates in pure Prolog. In *Proc. Int. Conf. on Fifth Generation Computer Systems*, pages 244–253, Tokyo, 1984.