Learning Modal Theories

J.W. Lloyd[1] and K.S. Ng[2]

[1]Computer Sciences Laboratory
Research School of Information Sciences and Engineering
The Australian National University
jwl@mail.rsise.anu.edu.au
[2]Symbolic Machine Learning and Knowledge Acquisition
National ICT Australia*
kee.siong@nicta.com.au

Abstract. This paper discusses how to learn theories that are modal, concentrating on the issue of how modal hypotheses are formed. Illustrations are given to show the usefulness of the ideas for agent applications.

1 Introduction

This paper introduces the idea of learning theories that are modal. To motivate the development, we first discuss why learning modal theories is useful, particularly in agent applications.

Consider an agent situated in some environment that can receive percepts from the environment and can apply actions to the environment. Included in a state of the agent may be information about the environment or something that is internal to the agent. The state may be updated as a result of receiving a percept. As well as some state, the agent's model includes its belief base, which can also be updated. Each action changes the current state to a new state. The agent selects an action that maximises its expected performance. An agent architecture based on the rationality principle of choosing an action that maximises expected utility is in [1] and discussion of the learning component of such agents is in [2].

We now concentrate on action selection. Agents use their belief bases to determine which action to select. It is common for the beliefs that are needed for this to have a modal nature, usually temporal or epistemic. For example, on the temporal side, it might be important that at the last time or at some time in the past, some situation held and, therefore, a certain action is now appropriate. Similarly, on the epistemic side, beliefs about the beliefs of other agents may be used to determine which action to perform. The usefulness of modal beliefs for agents is now well established, in [3] and [4], for example. Besides, introspection reveals that people use temporal and epistemic considerations when deciding what to do; essentially, we are exploiting here the fact that modal logic is a part

* NICTA is funded through the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

of mathematics which is useful for building agents that aspire to have similar capabilities.

While many beliefs can be built into agents beforehand by their designers, it is also common for beliefs to be acquired by some kind of learning process during deployment. Since beliefs can be modal, the hypothesis languages used by the learning system need to be modal. We are thus led to the conclusion that symbolic machine learning needs to be generalised beyond classical logics, such as first-order logic, to modal logics. In fact, modal higher-order logic will be employed in this paper.

This paper investigates the potential usefulness of modalities for learning applications. Its two main contributions are machinery for specifying modal hypotheses and illustrations that show the usefulness of modal hypotheses in agent applications. Given the generality of the agent paradigm and the ubiquity of agent applications, we believe that agents will be a fertile application area for symbolic machine learning techniques.

The next section contains a discussion of the logical machinery needed to construct modal hypotheses. Section 3 contains two illustrations of the ideas for agent applications. Section 4 gives some conclusions and discusses related work.

2 Modal Hypotheses

An approach to symbolic learning based on higher-order logic is presented in [5] that introduces the concept of a predicate rewrite system which is a grammar formalism for specifying search spaces of predicates that are used in hypothesis languages. Thus, to achieve the desired generalisation to learning *modal* theories, a key step is to extend predicate rewrite systems to the modal case. This is done in this section. Along the way, we introduce a modal, higher-order logic which provides a suitable setting for the development.

2.1 Modal Higher-order Logic

We outline the most relevant aspects of the logic, focussing to begin with on the monomorphic version. We define types and terms, and give an introduction to the modalities that will be most useful in this paper. Full details of the logic, including its reasoning capabilities, can be found in [6].

Definition 1. *An* alphabet *consists of three sets:*

1. *A set $\mathfrak{T}$ of type constructors.*
2. *A set $\mathfrak{C}$ of constants.*
3. *A set $\mathfrak{V}$ of variables.*

Each type constructor in $\mathfrak{T}$ has an arity. The set $\mathfrak{T}$ always includes the type constructor Ω of arity 0. Ω is the type of the booleans. Each constant in $\mathfrak{C}$ has a signature. The set $\mathfrak{V}$ is denumerable. Variables are typically denoted by $x, y, z, \ldots$. Types are built up from the set of type constructors, using the symbols $\rightarrow$ and $\times$.

Definition 2. *A* type *is defined inductively as follows.*

1. *If T is a type constructor of arity k and $\alpha_1, \ldots, \alpha_k$ are types, then $T\ \alpha_1 \ldots \alpha_k$ is a type. (Thus a type constructor of arity 0 is a type.)*
2. *If α and β are types, then $\alpha \rightarrow \beta$ is a type.*
3. *If $\alpha_1, \ldots, \alpha_n$ are types, then $\alpha_1 \times \cdots \times \alpha_n$ is a type.*

The set $\mathfrak{C}$ always includes the following constants.

1. $\top$ and $\bot$, having signature Ω.
2. $=_\alpha$, having signature $\alpha \rightarrow \alpha \rightarrow \Omega$, for each type α.
3. $\neg$, having signature $\Omega \rightarrow \Omega$.
4. $\wedge$, $\vee$, $\longrightarrow$, $\longleftarrow$, and $\longleftrightarrow$, having signature $\Omega \rightarrow \Omega \rightarrow \Omega$.
5. Σ_α and Π_α, having signature $(\alpha \rightarrow \Omega) \rightarrow \Omega$, for each type α.

The intended meaning of $=_\alpha$ is identity (that is, $=_\alpha x\ y$ is $\top$ iff x and y are identical), the intended meaning of $\top$ is true, the intended meaning of $\bot$ is false, and the intended meanings of the connectives $\neg$, $\wedge$, $\vee$, $\longrightarrow$, $\longleftarrow$, and $\longleftrightarrow$ are as usual. The intended meanings of Σ_α and Π_α are that Σ_α maps a predicate to $\top$ iff the predicate maps at least one element to $\top$ and Π_α maps a predicate to $\top$ iff the predicate maps all elements to $\top$. The type $\{\alpha\}$ is a synonym for $\alpha \rightarrow \Omega$, used when we are intuitively thinking of a term as a set of elements rather than as a predicate.

We assume there are necessity modality operators $\Box_i$, for $i = 1, \ldots, m$.

Definition 3. *A* term*, together with its type, is defined inductively as follows.*

1. *A variable in $\mathfrak{V}$ of type α is a term of type α.*
2. *A constant in $\mathfrak{C}$ having signature α is a term of type α.*
3. *If t is a term of type β and x a variable of type α, then $\lambda x.t$ is a term of type $\alpha \rightarrow \beta$.*
4. *If s is a term of type $\alpha \rightarrow \beta$ and t a term of type α, then $(s\ t)$ is a term of type β.*
5. *If $t_1, \ldots, t_n$ are terms of type $\alpha_1, \ldots, \alpha_n$, respectively, then $(t_1, \ldots, t_n)$ is a term of type $\alpha_1 \times \cdots \times \alpha_n$.*
6. *If t is a term of type α and $i \in \{1, \ldots, m\}$, then $\Box_i t$ is a term of type α.*

Terms of the form $(\Sigma_\alpha\ \lambda x.t)$ are written as $\exists_\alpha x.t$ and terms of the form $(\Pi_\alpha\ \lambda x.t)$ are written as $\forall_\alpha x.t$ (in accord with the intended meaning of Σ_α and Π_α). Thus, in higher-order logic, each quantifier is obtained as a combination of an abstraction acted on by a suitable function (Σ_α or Π_α).

If α is a type, then $\mathfrak{B}_\alpha$ is the set of basic terms of type α [5]. Basic terms represent individuals. For example, $\mathfrak{B}_\Omega$ is $\{\top, \bot\}$.

The polymorphic version of the logic extends what is given above by also having available parameters which are type variables (denoted by $a, b, c, \ldots$). The definition of a type as above is then extended to polymorphic types that may contain parameters and the definition of a term as above is extended to terms that may have polymorphic types. We work in the polymorphic version of the logic in the remainder of the paper. In this case, we drop the α in $\exists_\alpha$, $\forall_\alpha$, and $=_\alpha$, since the types associated with $\exists$, $\forall$, and $=$ are now inferred from the context.

An important feature of higher-order logic is that it admits functions that can take other functions as arguments. (First-order logic does not admit these so-called higher-order functions.) This fact can be exploited in applications, through the use of predicates to represent sets and predicate rewrite systems that are used for learning, for example.

The reasoning system employed by the learner combines a theorem prover and an equational reasoning system. The theorem prover is a fairly conventional tableau theorem prover for modal higher-order logic. The equational reasoning system is, in effect, a computational system that significantly extends existing functional programming languages by adding facilities for computing with modalities. The proof component and the computational component are tightly integrated, in the sense that either can call the other. Furthermore, this synergy between the two makes possible all kinds of interesting reasoning tasks. It turns out that, for agent applications, the most common reasoning task is a computational one, that of evaluating a function call. In this case, the theorem-prover plays a subsidiary role, usually that of performing some rather straightforward modal theorem-proving tasks.

We remark that the treatment of modalities in a computation has to be carefully handled. The reason is that even such a simple concept as applying a substitution is greatly complicated in the modal setting by the fact that constants generally have different meanings in different worlds and therefore the act of applying a substitution may not result in a term with the desired meaning. A similar problem occurs when the redex chosen for a computation step is in the scope of a modality. A standard way to handle these problems is to insist that some constants be *rigid*, that is, have the same meaning in each world (in the semantics). In the modal higher-order logic setting, it is entirely natural for some constants to be rigid; for example, all constants (data constructors and functions alike) in the Haskell prelude can be declared to be rigid, except in the most sophisticated applications. For non-rigid constants, of which there are usually many in the belief bases of typical agents, great care must be taken to ensure that they are only ever used in the correct modal contexts.

Theories in the logic consist of two kinds of assumptions, global and local. The essential difference is that global assumptions are true in each world in the intended interpretation, while local assumptions only have to be true in the actual world in the intended interpretation. Each kind of assumption has a certain role to play when proving a theorem.

As is well known, modalities can have a variety of meanings, depending on the application. Some of these are indicated here; much more detail can be found in [3], [4] and [6], for example.

In multi-agent applications, one meaning for $\Box_i\varphi$ is that 'agent i knows φ'. In this case, the modality $\Box_i$ is written as $\boldsymbol{K}_i$. The logic $\mathbf{S5}_m$ is commonly used to capture the intended meaning of knowledge.

A weaker notion is that of belief. In this case, $\Box_i \varphi$ means that 'agent i believes φ' and the modality $\Box_i$ is written as $\boldsymbol{B}_i$. The logic $\mathbf{KD45}_m$ is commonly used to capture the intended meaning of belief.

The modalities also have a variety of temporal readings. We will make use of the (past) temporal modalities $\CIRCLE$ ('last') and $\blacksquare$ ('always in the past'). We can also define the modality $\blacklozenge$ ('sometime in the past'), which is dual to $\blacksquare$, by $\blacklozenge t \equiv \neg \blacksquare \neg t$, where t is either a formula or a predicate. (The negation of a predicate is defined below.)

Modalities can be applied to terms that are not formulas. Thus terms such as $\boldsymbol{B}_i 42$ and $\CIRCLE A$, where A is a constant, are admitted. We will find to be particularly useful terms that have the form $\Box_{j_1} \cdots \Box_{j_r} f$, where f is a function and $\Box_{j_1} \cdots \Box_{j_r}$ is a sequence of modalities.

For a particular agent in some application, the *belief base* of the agent is a theory. There are no restrictions placed on theories. Each assumption in a belief base is called a *belief*. Typically, for agent j, local assumptions in its belief base have the form $\boldsymbol{B}_j \varphi$, with the intuitive meaning 'agent j believes φ'. Often φ is an equation. Other typical local assumptions have the form $\boldsymbol{B}_j \boldsymbol{B}_i \varphi$, meaning 'agent j believes that agent i believes φ'. Global assumptions in a belief base typically have the form φ, with no modalities at the front since the fact that they are global implicitly implies any sequence of (necessity) modalities effectively appears at the front. Thus, in general, beliefs commonly have the form $\boldsymbol{B}_{j_1} \cdots \boldsymbol{B}_{j_r} \varphi$, where $r \geq 0$. If there is a temporal component to beliefs, this is often manifested by temporal modalities at the front of beliefs. Then, for example, there could be a belief of the form $\CIRCLE^2 \boldsymbol{B}_j \boldsymbol{B}_i \varphi$, whose intuitive meaning is 'at the second last time, agent j believed that agent i believed φ'. (Here, $\CIRCLE^2$ is a shorthand for $\CIRCLE\CIRCLE$.)

The following schema can be used as a global assumption.

$$(\Box_i \mathbf{s}\ \mathbf{t}) = \Box_i (\mathbf{s}\ \mathbf{t}),$$

where $\mathbf{s}$ is a syntactical variable ranging over terms of type $\alpha \to \beta$ and $\mathbf{t}$ is a syntactical variable ranging over *rigid* terms of type α. (A term is rigid iff every constant in it is rigid.) This schema also holds for the dual modality $\blacklozenge$ (when β is Ω). Thus, under the rigidity assumption on $\mathbf{t}$, the schemas

$$(\boldsymbol{B}_i \mathbf{s}\ \mathbf{t}) = \boldsymbol{B}_i (\mathbf{s}\ \mathbf{t})$$
$$(\blacklozenge \mathbf{s}\ \mathbf{t}) = \blacklozenge (\mathbf{s}\ \mathbf{t})$$

are global assumptions. Assumptions like these are often used in evaluating predicates generated by predicate rewrite systems.

2.2 Predicate Rewrite Systems

In this subsection, we extend the predicate rewrite systems defined in [5] to the modal case. Predicates are built up by composing basic functions called transformations. Composition is handled by the (reverse) composition function

$$\circ : (a \to b) \to (b \to c) \to (a \to c)$$

defined by $((f \circ g)\ x) = (g\ (f\ x))$.

Definition 4. *A* transformation f *is a function having a signature of the form*

$$f : (\varrho_1 \to \Omega) \to \cdots \to (\varrho_k \to \Omega) \to \mu \to \sigma,$$

where any parameters in $\varrho_1, \ldots, \varrho_k$ and σ appear in μ, and $k \geq 0$. The type σ is called the target *of the transformation. The number k is called the* rank *of the transformation.*

Example 1. The transformation $\wedge_n : (a \to \Omega) \to \cdots \to (a \to \Omega) \to a \to \Omega$ defined by $\wedge_n\ p_1 \ldots p_n\ x = (p_1\ x) \wedge \cdots \wedge (p_n\ x)$, where $n \geq 2$, provides the 'conjunction' of n predicates. Disjunction ($\vee_n$) of predicates can be defined in a similar fashion.

The transformation $\neg : (a \to \Omega) \to a \to \Omega$ defined by

$$\neg p\ x = \neg(p\ x),$$

provides the negation of a predicate.

Consider the transformation $\mathit{setExists_1} : (a \to \Omega) \to \{a\} \to \Omega$ defined by

$$\mathit{setExists_1}\ p\ t = \exists x.((p\ x) \wedge (x \in t)).$$

The function ($\mathit{setExists_1}\ p$) checks whether a set has an element that satisfies p.

The transformation $\mathit{top} : a \to \Omega$ is defined by $\mathit{top}\ x = \top$, for each x. The transformation $\mathit{bottom} : a \to \Omega$ is defined by $\mathit{bottom}\ x = \bot$, for each x.

Many more transformations are given in [5].

Next the definition of the class of predicates formed by composing transformations is presented. In the following definition, it is assumed that some (possibly infinite) class of transformations is given and all transformations considered are taken from this class. A standard predicate is defined by induction on the number of (occurrences of) transformations it contains as follows. Let $\Box$ denote a (possibly empty) sequence of modalities $\Box_{j_1} \cdots \Box_{j_r}$.

Definition 5. *A* standard predicate *is a term of the form*

$$\Box_1(f_1\ p_{1,1} \ldots p_{1,k_1}) \circ \cdots \circ \Box_n(f_n\ p_{n,1} \ldots p_{n,k_n}),$$

where f_i is a transformation of rank k_i ($i = 1, \ldots, n$), the target of f_n is Ω, $\Box_i$ is a sequence of modalities ($i = 1, \ldots, n$), p_{i,j_i} is a standard predicate ($i = 1, \ldots, n$, $j_i = 1, \ldots, k_i$), $k_i \geq 0$ ($i = 1, \ldots, n$) and $n \geq 1$.

Definition 5 extends that of a (non-modal) standard predicate in [5] precisely in that the definition here allows modalities to appear.

Example 2. Let p and q be transformations of type $\sigma \to \Omega$. Then

$$\boldsymbol{B}_i(\mathit{setExists_1}\ (\wedge_2\ \bullet\boldsymbol{B}_j p\ \blacklozenge\boldsymbol{B}_j q))$$

is a standard predicate of type $\{\sigma\} \to \Omega$. If t is a (rigid) set of elements of type σ, then

$$(\boldsymbol{B}_i(\mathit{setExists_1}\ (\wedge_2\ \bullet\boldsymbol{B}_j p\ \blacklozenge\boldsymbol{B}_j q))\ \ t)$$

simplifies to

$$\boldsymbol{B}_i \exists x.((\bullet \boldsymbol{B}_j (p\ x) \wedge \blacklozenge \boldsymbol{B}_j (q\ x)) \wedge (x \in t)),$$

which is true iff agent i believes that there is an element x in t satisfying the property that at the last time agent j believed that x satisfied p and at some time in the past agent j believed that x satisfied q.

Now we can informally define a predicate rewrite system. A predicate rewrite is an expression of the form $p \rightarrowtail q$, where p and q are standard predicates. The predicate p is called the *head* and q is the *body* of the rewrite. A predicate rewrite system is a finite set of predicate rewrites. One should think of a predicate rewrite system as a kind of grammar for generating a particular class of predicates. Roughly speaking, this works as follows. Starting from the weakest predicate *top*, all predicate rewrites that have *top* (of the appropriate type) in the head are selected to make up child predicates that consist of the bodies of these predicate rewrites. Then, for each child predicate and each redex in that predicate, all child predicates are generated by replacing each redex by the body of the predicate rewrite whose head is identical to the redex. This generation of predicates continues to produce the entire space of predicates given by the predicate rewrite system. The details of the (non-modal) version of this can be found in [5]; the modal version works in a similar fashion.

Example 3. Consider the following predicate rewrite system.

$$\begin{aligned}
&top \rightarrowtail \boldsymbol{B}_i(setExists_1\ (\wedge_2\ top\ top))\\
&top \rightarrowtail \bullet \boldsymbol{B}_j\, top\\
&top \rightarrowtail \blacklozenge \boldsymbol{B}_j\, top\\
&top \rightarrowtail p\\
&top \rightarrowtail q\\
&top \rightarrowtail r.
\end{aligned}$$

The following is a path in the predicate space defined by the rewrite system.

$$\begin{aligned}
top \rightsquigarrow\ & \boldsymbol{B}_i(setExists_1\ (\wedge_2\ top\ top)) \rightsquigarrow \boldsymbol{B}_i(setExists_1\ (\wedge_2\ \bullet \boldsymbol{B}_j\, top\ \ top))\\
&\rightsquigarrow \boldsymbol{B}_i(setExists_1\ (\wedge_2\ \bullet \boldsymbol{B}_j p\ \ top))\\
&\quad \rightsquigarrow \cdots \rightsquigarrow \boldsymbol{B}_i(setExists_1\ (\wedge_2\ \bullet \boldsymbol{B}_j p\ \ \blacklozenge \boldsymbol{B}_j q)).
\end{aligned}$$

The set $P_{\rightarrowtail}$ of predicates that can be generated from a predicate rewrite system $\rightarrowtail$ is called a *predicate language*. Given some predicate language, it remains to specify the *hypothesis language*, that is, the form of learned theories that employ predicates in the predicate language. There are many possibilities. For the purpose of this paper, we can restrict attention to the class of decision lists [7] that can be formed. Each internal node in such a decision list would be made up of a predicate in the predicate language. For learning, we can employ standard rule-learning algorithms.

3 Illustrations

This section contains two illustrations of the usefulness of learning modal theories for agent applications.

3.1 Majordomo Agent

Consider a majordomo agent that manages a household. There are many tasks for such an agent to carry out including keeping track of occupants, turning appliances on and off, ordering food for the refrigerator, and so on.

Here we concentrate on one small aspect of the majordomo's tasks which is to recommend television programs for viewing by the occupants of the house. (See `http://www.netflixprize.com` for a related industrial problem.) Suppose the current occupants are Alice, Bob, and Cathy, and that the agent knows the television preferences of each of them. Methods for learning these preferences were studied in [2]. Suppose that each occupant has a personal agent that has learned (amongst many other functions) the function $\mathit{likes} : \mathit{Program} \rightarrow \mathit{\Omega}$, where *likes* is true for a program iff the person likes the program. We also suppose that the majordomo has access to the definitions of this function for each occupant, for the present time and for some suitable period into the past. Let $\boldsymbol{B}_m$ be the belief modality for the majordomo agent, $\boldsymbol{B}_a$ the belief modality for Alice, $\boldsymbol{B}_b$ the belief modality for Bob, and $\boldsymbol{B}_c$ the belief modality for Cathy. Thus part of the majordomo's belief base has the following form:

$$\begin{array}{l}
\boldsymbol{B}_m\boldsymbol{B}_a\ \forall x.((\mathit{likes}\ x) = \varphi_0)\\
\bullet\boldsymbol{B}_m\boldsymbol{B}_a\ \forall x.((\mathit{likes}\ x) = \varphi_1)\\
\qquad\vdots\\
\bullet^{n-1}\boldsymbol{B}_m\boldsymbol{B}_a\ \forall x.((\mathit{likes}\ x) = \varphi_{n-1})\\
\bullet^{n}\boldsymbol{B}_m\forall x.(\blacklozenge\boldsymbol{B}_a(\mathit{likes}\ x) = \bot)\\[1ex]
\boldsymbol{B}_m\boldsymbol{B}_b\ \forall x.((\mathit{likes}\ x) = \psi_0)\\
\bullet\boldsymbol{B}_m\boldsymbol{B}_b\ \forall x.((\mathit{likes}\ x) = \psi_1)\\
\qquad\vdots\\
\bullet^{k-1}\boldsymbol{B}_m\boldsymbol{B}_b\ \forall x.((\mathit{likes}\ x) = \psi_{k-1})\\
\bullet^{k}\boldsymbol{B}_m\forall x.(\blacklozenge\boldsymbol{B}_b(\mathit{likes}\ x) = \bot)\\[1ex]
\boldsymbol{B}_m\boldsymbol{B}_c\ \forall x.((\mathit{likes}\ x) = \xi_0)\\
\bullet\boldsymbol{B}_m\boldsymbol{B}_c\ \forall x.((\mathit{likes}\ x) = \xi_1)\\
\qquad\vdots\\
\bullet^{l-1}\boldsymbol{B}_m\boldsymbol{B}_c\ \forall x.((\mathit{likes}\ x) = \xi_{l-1})\\
\bullet^{l}\boldsymbol{B}_m\forall x.(\blacklozenge\boldsymbol{B}_c(\mathit{likes}\ x) = \bot),
\end{array}$$

for suitable φ_i, ψ_i, and ξ_i. The form these can take is explained in [2].

In the beginning, the belief base contains the formula

$$\boldsymbol{B}_m \forall x.(\blacklozenge \boldsymbol{B}_a(\mathit{likes}\ x) = \bot),$$

whose purpose is to prevent runaway computations into the infinite past for certain formulas of the form $\blacklozenge\varphi$. The meaning of this formula is "the agent believes that for all programs it is not true that at some time in the past Alice likes the program". After n time steps, this formula has been transformed into

$$\bullet^n \boldsymbol{B}_m \forall x.(\blacklozenge \boldsymbol{B}_a(\mathit{likes}\ x) = \bot).$$

In general, at each time step, the beliefs about *likes* at the previous time steps each have another $\bullet$ placed at their front to push them one step further back into the past, and a new current belief about *likes* is acquired.

Based on these beliefs about the occupant preferences for TV programs, the task for the agent is to recommend programs that all three occupants would be interested in watching together. The simplest idea is that the agent should only recommend programs that all three occupants currently like. But it is possible that less stringent conditions might also be acceptable; for example, it might be sufficient that two of the occupants currently like a program but that the third has liked the program in the past (even if they do not like it at the present time). Here is a (simplified) predicate rewrite system suitable for such a learning task.

$$\begin{aligned}
&\mathit{top} \rightarrowtail \wedge_3\ \mathit{top}\ \mathit{top}\ \mathit{top}\\
&\mathit{top} \rightarrowtail \vee_2\ \mathit{top}\ \mathit{top}\\
&\mathit{top} \rightarrowtail \boldsymbol{B}_i\mathit{likes} \quad \%\ \text{for each } i \in \{a,b,c\}\\
&\mathit{top} \rightarrowtail \blacklozenge\boldsymbol{B}_i\mathit{likes} \quad \%\ \text{for each } i \in \{a,b,c\}.
\end{aligned}$$

Let $\mathit{group\_likes} : \mathit{Program} \rightarrow \Omega$ be the function that the agent needs to learn. Thus the informal meaning of *group_likes* is that it is true for a program iff the occupants collectively like the program. (This may involve a degree of compromise by some of the occupants.) Training examples for this task look like

$$\begin{aligned}
&\boldsymbol{B}_m((\mathit{group\_likes}\ P_1) = \top)\\
&\boldsymbol{B}_m((\mathit{group\_likes}\ P_2) = \bot),
\end{aligned}$$

where P_1 and P_2 are particular programs. The definition of a typical function that might be learned from training examples and the hypothesis language given by the above predicate rewrite system is as follows.

$$\begin{aligned}
&\boldsymbol{B}_m\forall x.\,((\mathit{group\_likes}\ x) =\\
&\qquad \mathit{if}\ ((\wedge_3\ \blacklozenge\boldsymbol{B}_a\mathit{likes}\ \boldsymbol{B}_b\mathit{likes}\ \boldsymbol{B}_c\mathit{likes})\ x)\ \mathit{then}\ \top\\
&\qquad \mathit{else\ if}\ ((\wedge_3\ \boldsymbol{B}_c\mathit{likes}\ (\vee_2\ \boldsymbol{B}_a\mathit{likes}\ \boldsymbol{B}_b\mathit{likes})\ \mathit{top})\ x)\ \mathit{then}\ \top\\
&\qquad \mathit{else}\ \bot).
\end{aligned}$$

Now let P be some specific program. In Figure 1, we show the computation of $(\mathit{group\_likes}\ P)$. The redex selected is underlined at each step in the computation. The computation makes use of standard boolean functions defined in [5, Chap. 5] and axiom schemas like $\bullet \boldsymbol{B}_i\, \varphi \longrightarrow \boldsymbol{B}_i \bullet \varphi$ and $\blacklozenge \varphi = \varphi \vee \bullet \blacklozenge\, \varphi$. The former is used to prove that formulas of the form

$$\boldsymbol{B}_m \bullet^i \boldsymbol{B}_a\ \forall x.((\mathit{likes}\ x) = \varphi_i)$$

are theorems of the belief base. These theorems are then used to simplify the $(\mathit{likes}\, P)$ terms located in different modal contexts in the computation. It follows from the computation shown in Figure 1 that $\boldsymbol{B}_m((\mathit{group\_likes}\ P) = \bot)$ is a consequence of the belief base of the agent. On this basis, the agent will presumably not recommend to the occupants that they watch program P together.

In practice, one would use a richer hypothesis language for this problem. For example, the majordomo can also make use of beliefs held by the personal diary agents of Alice, Bob and Cathy in the hypothesis language. To recommend a program for common viewing, it is important, for example, that all three are free at the program time slot. Other relevant information can be included.

3.2 Learning by Revising Past Beliefs

For agents, learning is usually a continual life-long affair. For example, a recommender agent for television programs needs to track the changing preferences of its user over a life time. Similarly, to achieve optimal performance, an adaptive traffic-light control agent needs to monitor the traffic at regular intervals to keep its beliefs about current conditions updated. This section presents a general framework for incremental belief revision.

We will start by considering the following simplified form of the general problem. We want to track a function $f : \sigma \to \tau$ that changes slowly over time. We have access to the previous acquired definition $\bullet \boldsymbol{B}\,(f = \lambda x.\varphi)$ in the belief base. ($\boldsymbol{B}$ is the belief modality of the relevant agent.) A new training set arrives and now a new definition for f needs to be acquired. How do we proceed?

Obviously, in computing the current definition for f, we would like to reuse those parts of the previous definition that are still valid in the light of new evidence. One way to achieve that is to define an hypothesis language that captures the different ways the old definition can be changed, or perturbed, in small ways. We will show in stages how this can be done, starting with the description of a variant of the standard decision-list learning algorithm that will be needed.

The standard decision-list algorithm is a greedy algorithm. A set of examples is covered at every step, and an element of $\mathfrak{B}_\tau$ is used to label the leaf node constructed, the exact choice being determined by the majority class of the covered examples. This is equivalent to using a constant function to make predictions in the covered subregion. We extend the algorithm to use more complex functions for this purpose. In the new algorithm, a *label language* L is specified, in addition to a predicate language P. Learning proceeds via greedy

$$\underline{(\mathit{group\_likes}\ P)}$$
$$\mathit{if}\ \underline{((\wedge_3\ \blacklozenge\boldsymbol{B}_a\mathit{likes}\ \boldsymbol{B}_b\mathit{likes}\ \boldsymbol{B}_c\mathit{likes})\ P)}\ \mathit{then}\ \top\ \mathit{else}\ \ldots$$
$$\mathit{if}\ \underline{(\blacklozenge\boldsymbol{B}_a\mathit{likes}\ P)} \wedge (\boldsymbol{B}_b\mathit{likes}\ P) \wedge (\boldsymbol{B}_c\,\mathit{likes}\ P)\ \mathit{then}\ \top\ \mathit{else}\ \ldots$$
$$\mathit{if}\ \blacklozenge\underline{(\boldsymbol{B}_a\mathit{likes}\ P)} \wedge (\boldsymbol{B}_b\mathit{likes}\ P) \wedge (\boldsymbol{B}_c\mathit{likes}\ P)\ \mathit{then}\ \top\ \mathit{else}\ \ldots$$
$$\mathit{if}\ \underline{\blacklozenge\boldsymbol{B}_a(\mathit{likes}\ P)} \wedge (\boldsymbol{B}_b\mathit{likes}\ P) \wedge (\boldsymbol{B}_c\mathit{likes}\ P)\ \mathit{then}\ \top\ \mathit{else}\ \ldots$$
$$\mathit{if}\ (\boldsymbol{B}_a\underline{(\mathit{likes}\ P)} \vee \bullet\blacklozenge\boldsymbol{B}_a(\mathit{likes}\ P)) \wedge (\boldsymbol{B}_b\mathit{likes}\ P) \wedge (\boldsymbol{B}_c\mathit{likes}\ P)\ \mathit{then}\ \top\ \mathit{else}\ \ldots$$
$$\vdots$$
$$\mathit{if}\ (\underline{\boldsymbol{B}_a\bot} \vee \bullet\blacklozenge\boldsymbol{B}_a(\mathit{likes}\ P)) \wedge (\boldsymbol{B}_b\mathit{likes}\ P) \wedge (\boldsymbol{B}_c\mathit{likes}\ P)\ \mathit{then}\ \top\ \mathit{else}\ \ldots$$
$$\mathit{if}\ \underline{(\bot \vee \bullet\blacklozenge\boldsymbol{B}_a(\mathit{likes}\ P))} \wedge (\boldsymbol{B}_b\mathit{likes}\ P) \wedge (\boldsymbol{B}_c\mathit{likes}\ P)\ \mathit{then}\ \top\ \mathit{else}\ \ldots$$
$$\mathit{if}\ \bullet\underline{\blacklozenge\boldsymbol{B}_a(\mathit{likes}\ P)} \wedge (\boldsymbol{B}_b\mathit{likes}\ P) \wedge (\boldsymbol{B}_c\mathit{likes}\ P)\ \mathit{then}\ \top\ \mathit{else}\ \ldots$$
$$\mathit{if}\ \underline{\bullet(\boldsymbol{B}_a(\mathit{likes}\ P) \vee \bullet\blacklozenge\boldsymbol{B}_a(\mathit{likes}\ P))} \wedge (\boldsymbol{B}_b\mathit{likes}\ P) \wedge (\boldsymbol{B}_c\mathit{likes}\ P)\ \mathit{then}\ \top\ \mathit{else}\ \ldots$$
$$\mathit{if}\ (\bullet\boldsymbol{B}_a\underline{(\mathit{likes}\ P)} \vee \bullet^2\blacklozenge\boldsymbol{B}_a(\mathit{likes}\ P)) \wedge (\boldsymbol{B}_b\mathit{likes}\ P) \wedge (\boldsymbol{B}_c\mathit{likes}\ P)\ \mathit{then}\ \top\ \mathit{else}\ \ldots$$
$$\vdots$$
$$\mathit{if}\ (\underline{\bullet\boldsymbol{B}_a\bot} \vee \bullet^2\blacklozenge\boldsymbol{B}_a(\mathit{likes}\ P)) \wedge (\boldsymbol{B}_b\mathit{likes}\ P) \wedge (\boldsymbol{B}_c\mathit{likes}\ P)\ \mathit{then}\ \top\ \mathit{else}\ \ldots$$
$$\mathit{if}\ (\underline{\boldsymbol{B}_a\bot} \vee \bullet^2\blacklozenge\boldsymbol{B}_a(\mathit{likes}\ P)) \wedge (\boldsymbol{B}_b\mathit{likes}\ P) \wedge (\boldsymbol{B}_c\mathit{likes}\ P)\ \mathit{then}\ \top\ \mathit{else}\ \ldots$$
$$\mathit{if}\ (\underline{\bot \vee \bullet^2\blacklozenge\boldsymbol{B}_a(\mathit{likes}\ P))} \wedge (\boldsymbol{B}_b\mathit{likes}\ P) \wedge (\boldsymbol{B}_c\mathit{likes}\ P)\ \mathit{then}\ \top\ \mathit{else}\ \ldots$$
$$\mathit{if}\ \bullet^2\underline{\blacklozenge\boldsymbol{B}_a(\mathit{likes}\ P)} \wedge (\boldsymbol{B}_b\mathit{likes}\ P) \wedge (\boldsymbol{B}_c\mathit{likes}\ P)\ \mathit{then}\ \top\ \mathit{else}\ \ldots$$
$$\vdots$$
$$\mathit{if}\ \bullet^n\underline{\blacklozenge\boldsymbol{B}_a(\mathit{likes}\ P)} \wedge (\boldsymbol{B}_b\mathit{likes}\ P) \wedge (\boldsymbol{B}_c\mathit{likes}\ P)\ \mathit{then}\ \top\ \mathit{else}\ \ldots$$
$$\mathit{if}\ \underline{\bullet^n\bot} \wedge (\boldsymbol{B}_b\mathit{likes}\ P) \wedge (\boldsymbol{B}_c\mathit{likes}\ P)\ \mathit{then}\ \top\ \mathit{else}\ \ldots$$
$$\vdots$$
$$\mathit{if}\ \underline{\bot \wedge (\boldsymbol{B}_b\mathit{likes}\ P)} \wedge (\boldsymbol{B}_c\mathit{likes}\ P)\ \mathit{then}\ \top\ \mathit{else}\ \ldots$$
$$\mathit{if}\ \underline{\bot \wedge (\boldsymbol{B}_c\mathit{likes}\ P)}\ \mathit{then}\ \top\ \mathit{else}\ \ldots$$
$$\underline{\mathit{if}\ \bot\ \mathit{then}\ \top\ \mathit{else}\ \ldots}$$
$$\vdots$$
$$\bot$$

Fig. 1. Computation using $\boldsymbol{B}_m$ of $(\mathit{group\_likes}\ P)$

search in the usual fashion. At every step, we seek $\arg\max_{p \in P} s(S_p)$, where S_p is the subset of the current set of examples covered by p and $s(S)$ is defined to be $\max_{l \in L} |\{(x, y) \in S : (l\ x) = y\}|$. The maximising label function l^* for the maximising predicate p^* is then used to label S_{p^*}. Accuracy is used as the heuristic function here; other measures can be used instead, of course.

We have described the algorithm. The next step is to define a suitable predicate language for use with it. In doing that, first we have to consider the structure of a decision list, which has the following general form:

$$\lambda x.\mathit{if}\ (p_1\ x)\ \mathit{then}\ v_1\ \mathit{else}\ \mathit{if}\ (p_2\ x)\ \mathit{then}\ v_2\ \ldots\ \mathit{else}\ \mathit{if}\ (p_n\ x)\ \mathit{then}\ v_n\ \mathit{else}\ v_0. \quad (1)$$

Writing q_{p_i} for $\wedge_i\ \neg p_1\ \ldots\ \neg p_{i-1}\ p_i$ (where $q_{p_1} = p_1$ in the base case), this term is equivalent to

$$\lambda x.\mathit{if}\ (q_{p_1}\ x)\ \mathit{then}\ v_1\ \mathit{else}\ \mathit{if}\ (q_{p_2}\ x)\ \mathit{then}\ v_2\ \ldots\ \mathit{else}\ \mathit{if}\ (q_{p_n}\ x)\ \mathit{then}\ v_n\ \mathit{else}\ v_0,$$

which we will call the *expanded form* of (1). We will effectively work with the expanded form of a decision list in designing a suitable predicate language. (This is done implicitly; expanded forms of decision lists are never explicitly constructed.)

We now proceed with the definition of a predicate language. The following transformation plays a key role.

$$\begin{aligned}
&\mathit{covered} : \mathit{Int} \to \mathit{Int} \to (a \to b) \to (a \to \Omega)\\
&\mathit{covered}\ i\ j\ \lambda x.\mathit{if}\ (p_1\ x)\ \mathit{then}\ v_1\ \mathit{else}\\
&\qquad\qquad \mathit{if}\ (p_2\ x)\ \mathit{then}\ v_2\ \ldots\ \mathit{else}\ \mathit{if}\ (p_n\ x)\ \mathit{then}\ v_n\ \mathit{else}\ v_0\\
&= \mathit{if}\ (i = 1)\ \mathit{then}\ (\vee_j\ p_1\ \ldots\ p_j)\ \mathit{else}\ (\wedge_i\ \neg p_1\ \ldots\ \neg p_{i-1}\ (\vee_{j-i+1}\ p_i\ \ldots\ p_j)).
\end{aligned}$$

Thus, given a decision list f, $((\mathit{covered}\ i\ j\ f)\ x)$ evaluates to true iff the individual x falls into one of the nodes between the ith and jth nodes inclusively. Let $\rightarrowtail$ be the original predicate rewrite system used to acquire the previous definition for f. The desired hypothesis predicate language is obtained by adding to $\rightarrowtail$ the following predicate rewrites:

$$\mathit{top} \rightarrowtail (\mathit{covered}\ i\ j\ \bullet f) \qquad \%\ \text{for each}\ i, j \in \{1, \ldots, N\}, i \le j,$$

where N is the number of nodes in the previous definition for f.

We have specified the predicate language $P_{\rightarrowtail}$. It remains to specify a suitable label language. For that, the set $\mathfrak{B}_\tau \cup \{ (\bullet f\ x) \}$ is adopted.

We now show that the space of functions defined by the given decision-list algorithm in conjunction with the specified predicate and label languages contains most of the ways we might want to modify an existing decision list. For convenience, we write $\langle (p_1, v_1), (p_2, v_2), \ldots, (p_n, v_n), (\mathit{top}, v_0) \rangle$ as a notational shorthand for a term having the form of (1) in the following. Suppose we have the following formula in the belief base:

$$\bullet \boldsymbol{B}\, (f = \langle (p_1, v_1), (p_2, v_2), \ldots, (p_{99}, v_{99}), (\mathit{top}, v_0) \rangle). \quad (2)$$

The following examples show how local surgery on (the expanded form of) the decision list can be realised using the hypothesis language defined. More complex operations can be achieved in a similar fashion.

Example 4. The operation of adding a node (r, v), where $r \in P_{\rightarrowtail}$, to the front of (2) can be realised by the definition

$$\boldsymbol{B}\,(f = \lambda x.\mathit{if}\ (r\ x)\ \mathit{then}\ v\ \mathit{else}\ (\bullet f\ x)).$$

Example 5. The operation of adding a node (r, v), where $r \in P_{\rightarrowtail}$, to the end of (2) can be realised by the definition

$$\boldsymbol{B}\,(f = \lambda x.\mathit{if}\ ((\mathit{covered}\ 1\ 99\ \bullet f)\ x)\ \mathit{then}\ (\bullet f\ x)\ \mathit{else}\ \mathit{if}\ (r\ x)\ \mathit{then}\ v\ \mathit{else}\ v_0),$$

which is equivalent to $\boldsymbol{B}\,(f = \langle (q_{p_1}, v_1), (q_{p_2}, v_2), \ldots, (q_{p_{99}}, v_{99}), (r, v), (\mathit{top}, v_0)\rangle)$.

Example 6. Consider the expanded form of (2). The operation of adding a node (r, v), where $r \in P_{\rightarrowtail}$, between $(q_{p_{29}}, v_{29})$ and $(q_{p_{30}}, v_{30})$ and removing the node $(q_{p_{77}}, v_{77})$ can be realised using

$$\begin{aligned}\boldsymbol{B}\,(f = \lambda x.\mathit{if}\ &((\mathit{covered}\ 1\ 29\ \bullet f)\ x)\ \mathit{then}\ (\bullet f\ x)\\ &\mathit{else}\ \mathit{if}\ (r\ x)\ \mathit{then}\ v\\ &\mathit{else}\ \mathit{if}\ ((\mathit{covered}\ 30\ 76\ \bullet f)\ x)\ \mathit{then}\ (\bullet f\ x)\\ &\mathit{else}\ \mathit{if}\ ((\mathit{covered}\ 78\ 99\ \bullet f)\ x)\ \mathit{then}\ (\bullet f\ x)\ \mathit{else}\ v_0),\end{aligned}$$

which can be unfolded into the following equivalent definition:

$$\begin{aligned}\boldsymbol{B}\,(f = \langle &(q_{p_1}, v_1), (q_{p_2}, v_2), \ldots, (q_{p_{29}}, v_{29}), (r, v), (q_{p_{30}}, v_{30}),\\ &(q_{p_{31}}, v_{31}), \ldots, (q_{p_{76}}, v_{76}), (q_{p_{78}}, v_{78}), \ldots, (q_{p_{99}}, v_{99}), (\mathit{top}, v_0)\rangle).\end{aligned}$$

Extensions to the Basic Setup We now consider some extensions to the basic setup. To begin with, we will record all past definitions for f in the belief base. Thus our belief base will contain, among other things, the following formulas:

$$\begin{aligned}&\bullet\,\boldsymbol{B}\,(f = \lambda x.\varphi_1)\\ &\vdots\\ &\bullet^{n-1}\,\boldsymbol{B}\,(f = \lambda x.\varphi_{n-1})\\ &\bullet^{n}\,\blacksquare\,\boldsymbol{B}\,(f = \lambda x.\varphi_n).\end{aligned}$$

We can add the following predicate rewrites to our rewrite system to pick out parts of any old definition previously learned.

$$\mathit{top} \rightarrowtail (\mathit{covered}\ j\ k\ \bullet^{i} f) \quad \text{\% for suitable values of } i, j \text{ and } k.$$

If desired, one can also enrich the predicate rewrite system with predicate rewrites that capture conditions that have occurred at least once in the past or in the recent past, or those that have always held in the past.

Example 7. Assume the function f changes in a cyclical manner. If we already have a good definition for each phase of the cycle, the algorithm should return

$$\boldsymbol{B}\,(f = \lambda x.\mathit{if}\ (\mathit{top}\ x)\ \mathit{then}\ (\bullet^i f\ x)\ \mathit{else}\ v_0),$$

for some i, as the current definition.

Example 8. We can piece together parts from definitions obtained at different times to form the current definition. For instance, we can have

$$\begin{aligned}\boldsymbol{B}\,(f = \lambda x.\mathit{if}\ &((\mathit{covered}\ 2\ 8\ \bullet^2 f)\ x)\ \mathit{then}\ (\bullet^2 f\ x)\\ &\mathit{else\ if}\ ((\mathit{covered}\ 6\ 9\ \bullet^4 f)\ x)\ \mathit{then}\ (\bullet^4 f\ x)\ \mathit{else}\ v_0).\end{aligned}$$

4 Conclusions

This paper has introduced some key ideas needed to learn theories that are modal. The first contribution is machinery for specifying modal hypothesis languages that extends the higher-order logic learning setting in [5]. Modalities have obvious usefulness as a language feature; the general setup introduced here shows a good way to incorporate them into the learning process. We would expect that the more traditional ILP settings [8] can be 'upgraded' in an analogous fashion.

The two illustrations given constitute the second contribution of this paper. Together they illustrate the kind of new possibilities opened up by having modalities in the hypothesis language. The multi-agent-learning paradigm exemplified by the majordomo agent is novel in ILP and has a lot of potential. The theory revision example provides a fresh perspective on an old ILP problem. Its relation to existing techniques is discussed below. A common thread that ties the two illustrations together is *learning from multiple sources of knowledge.*

The technologies introduced here are new and more work needs to be done. We have a prototype implementation of what is described here. The next step is to carry out substantial experiments to confirm the effectiveness of the approach. The complexity of learning modal theories can be analysed in the framework given in [9]. We expect results, both positive and negative, similar to those established in the non-modal setting to continue to hold in the modal setting. In other words, modalities do not come at a significant cost.

Related work Description logic can be regarded as a form of modal logic [4]. Related work can be found in the literature on learning theories in description logic. (See [10] and [11], for example, and the references therein.)

Incremental theory revision has long been studied in ILP following [12] and [13]. The framework introduced here allows the new definition to be obtained by revising previously acquired definitions going back multiple steps. Existing frameworks are restricted to the revision of *one* previous definition. The other noteworthy difference is that admissible revision operations are captured in the hypothesis language in our framework, *not* in the actual theory revision algorithm used as in existing setups.

There is an extensive literature on belief revision much of which was inspired by [14]. In these works, if modal logic is employed at all it is usually as a logical meta-language for the belief revision process itself, rather than the logic in which the beliefs are expressed (which is usually propositional). We are not aware of any works on belief revision in which the logic of the beliefs is as rich as modal higher-order logic. Also existing belief revision frameworks do not consider *generalisation*, which is a key aspect of learning and, we would argue, an essential component of any process by which a reasonably sophisticated agent might acquire new beliefs. On the other hand, we have not explicitly addressed here the important issue of inconsistency as frameworks for belief revision do.

References

1. Lloyd, J., Sears, T.: An architecture for rational agents. In Baldoni, M., *et al*, eds.: Declarative Agent Languages and Technologies (DALT 2005). Springer, LNAI 3904 (2006) 51–71
2. Cole, J., Gray, M., Lloyd, J., Ng, K.: Personalisation for user agents. In Dignum, F., *et al*, eds.: 4th Int. Conference on Autonomous Agents and Multiagent Systems (AAMAS 05). (2005) 603–610
3. Fagin, R., Halpern, J., Moses, Y., Vardi, M.: Reasoning about Knowledge. MIT Press (1995)
4. Gabbay, D., Kurucz, A., Wolter, F., Zakharyaschev, M.: Many-Dimensional Modal Logics: Theory and Applications. Studies in Logic and The Foundations of Mathematics, Volume 148. Elsevier (2003)
5. Lloyd, J.: Logic for Learning. Cognitive Technologies. Springer (2003)
6. Lloyd, J.: Knowledge representation and reasoning in modal higher-order logic. `http://csl.anu.edu.au/~jwl`, submitted for publication (2006)
7. Rivest, R.: Learning decision lists. Machine Learning **2** (1987) 229–246
8. De Raedt, L.: Logical settings for concept learning. Artificial Intelligence **95** (1997) 187–201
9. Ng, K.: (Agnostic) PAC learning concepts in higher-order logic. In: Proc. 17th European Conference on Machine Learning (ECML 2006). Springer, LNAI 4212 (2006) 711–718
10. Kietz, J.U.: Learnability of description logic programs. In Matwin, S., Sammut, C., eds.: Proc. 12th International Conference on Inductive Logic Programming (ILP 2002). Springer, LNCS 2583 (2003) 117–132
11. Badea, L., Nienhuys-Cheng, S.H.: A refinement operator for description logics. In Cussens, J., Frisch, A., eds.: Proc. 10th International Conference on Inductive Logic Programming (ILP 2000). Springer, LNAI 1866 (2000) 40–59
12. Sammut, C.: Learning Concepts by Performing Experiments. PhD thesis, University of New South Wales, Australia (1981)
13. De Raedt, L.: Interactive Theory Revision: An Inductive Logic Programming Approach. Academic Press (1992)
14. Alchourrón, C., Gärdenfors, P., Makinson, D.: On the logic of theory change: Partial meet contraction and revision functions. Journal of Symbolic Logic **50** (1985) 510–530