Reflections on Agent Beliefs

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

[1] College of Engineering and Computer Science
The Australian National University
jwl@cecs.anu.edu.au
[2] National ICT Australia
kee.siong@nicta.com.au

Abstract. Some issues concerning beliefs of agents are discussed. These issues are the general syntactic form of beliefs, the logic underlying beliefs, acquiring beliefs, and reasoning with beliefs. The logical setting is more expressive and aspects of the reasoning and acquisition processes are more general than are usually considered.

1 Introduction

Beliefs are an important component of every agent system that assist in the selection of actions. Because of their importance, there is a huge literature on representing, reasoning with, and acquiring beliefs. This paper contributes to this literature with a setting for beliefs that employs an unusually expressive logic.

We argue that since the purpose of beliefs is to help select actions, the general syntactic form for beliefs matters and that this form should be function definitions. We also argue that it is desirable that the logic in which these definitions are written be as expressive as possible. For this reason, we admit higher-order functions so that functions may take other functions as arguments. This means that the programming idioms of functional programming are available, and that sets and multisets can be represented by abstractions. Also it is common for beliefs 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 [1] and [2], for example. Besides, introspection reveals that people use temporal and epistemic considerations when deciding what to do. These considerations lead to the choice of multi-modal, higher-order logic as the logic for the beliefs.

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. We discuss an approach to belief acquisition that includes as special cases simple updating, belief revision [3], and learning [4].

During action selection, it is necessary to reason about beliefs or, more accurately in our case, compute with beliefs. We discuss a computation system for the logic that greatly extends existing modal and temporal logic programming systems, and give examples to illustrate how computation works. For most applications, computation is efficient enough that it could be used to select actions in real time.

In summary, the main contribution of this paper is a setting for agent beliefs in an expressive logic. A computation system that forms the core of a modal functional logic programming language is provided to reason about beliefs. All the facilities described here have been implemented.

The next section contains a discussion of the necessary logical machinery. Section 3 motivates the idea that beliefs should be function definitions. Section 4 shows how agents can acquire beliefs. Section 5 discusses how reasoning with beliefs is handled. Section 6 gives some conclusions.

2 Logic

In this section, 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 [5].

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 $(\alpha \to \Omega) \to \Omega$, 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 $\top$.

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 $\alpha \to \beta$.*
4. *If s is a term of type $\alpha \to \beta$ 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).

Constants can be declared to be *rigid*; they then have the same meaning in each world (in the semantics). A term is *rigid* if every constant in it is rigid.

If α is a type, then $\mathfrak{B}_\alpha$ is the set of basic terms of type α [6]. Basic terms represent individuals. For example, $\mathfrak{B}_\Omega$ is $\{\top, \bot\}$. Also $\mathfrak{B}_{Int}$ is $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$.

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. The universal closure of a formula φ is denoted by $\forall(\varphi)$.

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.

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. A theory is denoted by a pair $(\mathcal{G}, \mathcal{L})$, where $\mathcal{G}$ is the set of global assumptions and $\mathcal{L}$ is the set of local assumptions.

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 [1], [2] and [5], 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 $\boldsymbol{K}_i$.

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 modalities also have a variety of temporal readings. We will make use of the (past) temporal modalities $\bullet$ ('last') and $\blacksquare$ ('always in the past'). We also use the modality $\blacklozenge$ ('sometime in the past'), which is dual to $\blacksquare$.

Modalities can be applied to terms that are not formulas. Thus terms such as $\boldsymbol{B}_i 42$ and $\bullet A$, 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.

Throughout, it is assumed that all belief bases contain the standard equality theory given in [5] which includes definitions for equality, the connectives, the quantifiers, the *if_then_else* function, an assumption that gives β-reduction, and some assumptions concerning modalities.

One of these modal assumptions is the following schema that 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. Specialised to some of the epistemic and temporal modalities discussed so far, this means, for example, that

$$(\boldsymbol{B}_i \mathbf{s}\ \mathbf{t}) = \boldsymbol{B}_i(\mathbf{s}\ \mathbf{t}) \quad \text{and} \quad (\bullet \mathbf{s}\ \mathbf{t}) = \bullet(\mathbf{s}\ \mathbf{t})$$

are global assumptions (under the rigidity assumption on $\mathbf{t}$).

Another useful global assumption in the standard equality theory is

$$\Box_i \boldsymbol{t} = \boldsymbol{t},$$

where $\boldsymbol{t}$ is a syntactical variable ranging over *rigid* terms and $i \in \{1, \ldots, m\}$. Instances of this schema that could be used as global assumptions include the following.

$$\boldsymbol{B}_i 42 = 42, \quad \boldsymbol{B}_i \top = \top \quad \text{and} \quad \bullet\bot = \bot.$$

Let $(X, \mathcal{A}, \mu)$ be a measure space. A *density* (with respect to the measure μ) is a non-negative, integrable function h on X such that $\int_X h\, d\mu = 1$. We let *Density* σ denote the type of densities defined on sets whose elements have type σ.

3 Beliefs as Function Definitions

In this section, we discuss suitable syntactic forms for beliefs. There are no generally agreed forms for beliefs in the literature, other than the basic requirement that they be formulas. For the purpose of constructing multi-agent systems, we propose the following definition.

Definition 4. *A* belief *is the definition of a function* $f : \sigma \to \tau$ *having the form*

$$\Box \forall x.((f\ x) = t),$$

where $\Box$ *is a (possibly empty) sequence of modalities and* t *is a term of type* τ*.*

A belief base *is a set of beliefs.*

Typically, for agent j, beliefs have the form $\boldsymbol{B}_j\varphi$, with the intuitive meaning 'agent j believes φ', where φ is $\forall x.((f\ x) = t)$. Other typical beliefs have the form $\boldsymbol{B}_j\boldsymbol{B}_i\varphi$, meaning 'agent j believes that agent i believes φ'. 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 $\bullet^2\boldsymbol{B}_j\boldsymbol{B}_i\varphi$, whose intuitive meaning is 'at the second last time, agent j believed that agent i believed φ'. (Here, $\bullet^2$ is a shorthand for $\bullet\bullet$.)

We will now use the rational agent architecture described in [7] to motivate the introduction of Definition 4 and illustrate its usefulness. The arguments used are sufficiently general to be applicable to more general (PO)MDP-based agent architectures. Consider an agent application for which σ is the type of (internal) states of the agent and α is the type of actions. The dynamics of the agent can thus be modelled with a density f : *Density* $\sigma \times \alpha \times \sigma$. By conditioning on the first two arguments, we get a (conditional) density $f' : \sigma \times \alpha \to$ *Density* σ. If the agent is in a certain state and a certain action is applied, this function gives a distribution over the states the agent could end up in as a result of applying that action. In principle, knowing this transition distribution and the utility of states is enough to make a rational choice of action, where rational means choosing the action with the maximum expected utility.

The problem is that, in practice, there are usually a very large number of states which makes the direct use of this approach infeasible. To reduce the difficulty, an obvious idea is to define features on the state space in order to partition it into (a much smaller number of) equivalence classes of states that can be treated uniformly. This idea is illustrated in Figure 1, where *initial* projects onto the initial state, *action* projects onto the action, and *final* projects onto the state that is reached as a result of applying that action. The features are the evidence features e_i, for $i = 1, \ldots, n$, and the result features r_j, for $j = 1, \ldots, m$. Each class of features serves a different purpose. The evidence features are chosen so as to assist the selection of a good action, whereas the result features are chosen so as to provide a good evaluation of the resulting state.

Now consider the function

$$\mathit{transition} : \epsilon_1 \times \cdots \times \epsilon_n \times \alpha \to \mathit{Density}\ \rho_1 \times \cdots \times \rho_m$$

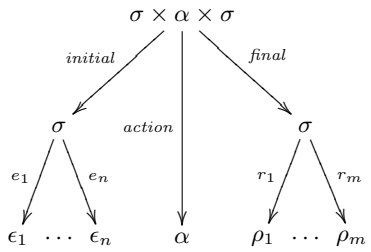

Fig. 1. Evidence and result features

that could be learned by an agent using training examples which indicate the state that results (possibly non-deterministically) from applying a particular action to a particular state. Given the function *transition*, the policy function $policy : \sigma \to \alpha$ is then defined by

$$(policy\ s) = \operatorname*{argmax}_a \mathbb{E}_{(transition\ ((e_1\ s),\ldots,(e_n\ s),a))}(utility),$$

where s ranges over states, a ranges over actions, and *utility* is a (real-valued) random variable over a product space of type $\rho_1 \times \cdots \times \rho_m$ that defines the utility of each tuple in this space. Here $\mathbb{E}_{(transition\ ((e_1\ s),\ldots,(e_n\ s),a))}$ denotes the expectation with respect to the density $(transition\ ((e_1\ s),\ldots,(e_n\ s),a))$.

Now consider this question: what makes up the belief base of such an agent? Clearly, the definitions of the evidence and (some of the) result features should be in the belief base. Further, the definitions of the functions *transition*, *utility* and *policy* should also be in the belief base. And these are all the beliefs the agent needs to maintain about the environment in order to act rationally. This concludes our motivation for Definition 4.

We now examine the form that beliefs can take in more detail. Some beliefs can be specified directly by the programmer and the body of the definition can be any term of the appropriate type. Some beliefs, however, need to be acquired from training examples, usually during deployment. We propose a particular form for beliefs of this latter type. We consider beliefs that, for a function $f : \sigma \to \tau$, are definitions of the following form.

$$\begin{aligned}
&\Box\forall x.((f\ x) = \\
&\quad \textit{if}\ (p_1\ x)\ \textit{then}\ v_1 \\
&\quad \textit{else if}\ (p_2\ x)\ \textit{then}\ v_2 \\
&\qquad \vdots \\
&\quad \textit{else if}\ (p_n\ x)\ \textit{then}\ v_n \\
&\quad \textit{else}\ v_0),
\end{aligned}$$

where $\Box$ is a (possibly empty) sequence of modalities, $p_1, \ldots, p_n$ are predicates that can be modal and/or higher order, and $v_0, v_1, \ldots, v_n$ are suitable values.

Such a belief is a definition for the function f in the context of the modal sequence $\Box$.

While the above form for beliefs may appear to be rather specialised, it turns out to be convenient and general, and easily encompasses beliefs in more conventional form. Here is an example to illustrate how one can represent a (relational) database.

Example 1. Consider an agent that recommends TV programs. Amongst other things the agent will need to have access to a TV guide as part of its belief base. Represented as a relational database, the TV guide would consist of a set of tuples, where each tuple gave details of the program that is on at a certain date, time, and channel. Similarly, as a Prolog program, the TV guide would be the corresponding set of facts. Actually, neither of these representations is a good one because each ignores a functional dependency in the data: each date, time and channel triple uniquely determines a program. Here we represent the TV guide as a function definition that correctly models this functional dependency.

For this, we require the following type synonyms.

$$\begin{aligned}
&\mathit{Occurrence} = \mathit{Date} \times \mathit{Time} \times \mathit{Channel}\\
&\mathit{Date} = \mathit{Day} \times \mathit{Month} \times \mathit{Year}\\
&\mathit{Time} = \mathit{Hour} \times \mathit{Minute}\\
&\mathit{Program} = \mathit{Title} \times \mathit{Duration} \times (\mathit{List}\ \mathit{Genre}) \times \mathit{Classification} \times \mathit{Synopsis}.
\end{aligned}$$

Now we can give (a typical instance of) the definition of the function

$$\mathit{tv\_guide} : \mathit{Occurrence} \to \mathit{Program}$$

that models the TV guide.

$$\begin{aligned}
&\boldsymbol{B}_t\ \forall x.((\mathit{tv_guide}\ x) =\\
&\quad \mathit{if}\ ((= ((21,7,2004),(19,30),\mathit{WIN}))\ x)\\
&\qquad\qquad \mathit{then}\ (\text{“}\mathit{Seinfeld}\text{”}, 30, [\mathit{Sitcom}], \mathit{PG}, \text{“}\mathit{Kramer}\ldots\text{”})\\
&\quad \mathit{else\ if}\ ((= ((20,7,2004),(20,30),\mathit{ABC}))\ x)\\
&\qquad\qquad \mathit{then}\ (\text{“}\mathit{The\ Bill}\text{”}, 50, [\mathit{Drama}], M, \text{“}\mathit{Sun\ Hill}\ldots\text{”})\\
&\qquad \vdots\\
&\quad \mathit{else}\ (\text{“ ”}, 0, [], \mathit{NA}, \text{“ ”})),
\end{aligned}$$

where $\boldsymbol{B}_t$ is the belief modality for the TV recommender and $(\text{“ ”}, 0, [], \mathit{NA}, \text{“ ”})$ is the default program (where 'default' has a technical meaning [6]). It is worth noting that all the queries that one might want to pose to the relational database (or Prolog) version of the TV guide can be just as easily posed to, and answered by, the function definition form (using computation, as discussed in Section 5).

It is also straightforward to rewrite Horn clause theories, a common way of representing beliefs, as function definitions in the form above.

Example 2. Consider an agent with belief modality $\boldsymbol{B}$ that has beliefs of the form

$$\begin{array}{l}\boldsymbol{B}((p\ t_1) \longleftarrow W_1) \\ \quad \vdots \\ \boldsymbol{B}((p\ t_n) \longleftarrow W_n).\end{array}$$

This form of belief base includes Horn clause theories and logic programs. By adding equations to the bodies and existentially quantifying free local variables in the bodies, the beliefs can be written in the form

$$\begin{array}{l}\boldsymbol{B}((p\ x) \longleftarrow V_1) \\ \quad \vdots \\ \boldsymbol{B}((p\ x) \longleftarrow V_n).\end{array}$$

This set of beliefs can then be written in the function definition form

$$\begin{array}{l}\boldsymbol{B}\ \forall x.((p\ x) = \\ \quad \mathit{if}\ (\lambda x.V_1\ x)\ \mathit{then}\ \top \\ \qquad \vdots \\ \quad \mathit{else\ if}\ (\lambda x.V_n\ x)\ \mathit{then}\ \top \\ \quad \mathit{else}\ \bot),\end{array}$$

which is equivalent to the original set of beliefs under the closed world assumption. (The latter formula is essentially the completion of the original set of beliefs, probably the semantics intended anyway.)

4 Acquiring Beliefs

Now we turn to belief acquisition. Belief bases are generally dynamic, that is, they change from time to time during deployment of the agent. It follows that agents need to have some method by which they can acquire new beliefs. We use the phrase 'belief acquisition' to name this process. The term 'acquire' is intended to be understood in a general sense that includes 'update', 'revise' and 'learn' as special cases. 'Update' refers to the simplest form of belief acquisition in which facts are added to or deleted from a simple database, 'revise' refers to the form of acquisition that is studied in the literature on belief revision [3], and 'learning' refers to machine learning [4]. Belief acquisition thus covers the spectrum from simple updating at one end to the generalisation that is characteristic of learning at the other end.

The approach we take to belief acquisition starts from the machine learning perspective in that it extends decision-list learning in [8]. In machine learning,

one wants to learn a function definition. The input to the learning process is a collection of training examples that give the value of the function for some points in its domain. A space of hypotheses is searched to find a definition for the function that agrees 'as well as possible' with the training examples, according to some measure. The hypothesis learned is intended to generalise, in the sense that it should give the correct value on unseen examples.

We extend the learning process in several ways so that it also includes update and belief revision. The first extension is that training examples can give the value of the function not just on a single point of the domain but on a subset of it given by some predicate. This allows us to capture some aspects of what happens in theory revision. In addition, the predicate can include modalities. Then, in order to control where on the spectrum from updating to learning we want to be, we make a careful choice of hypothesis language. If we want simple updating, then the hypothesis language is chosen to be very specific; if we want learning, then the hypothesis language is chosen to be general; for intermediate points on the spectrum, the hypothesis language is chosen accordingly.

A major ingredient for belief acquisition is a method of generating predicates. For this, we use predicate rewrite systems which we describe informally as follows. A predicate rewrite is an expression of the form $p \rightarrowtail q$, where p and q are predicates (in a particular syntactic form). 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* (defined below), 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 [6] and the modal version in [5].

A particular predicate language, called the basic language, often arises in applications.

Definition 5. *Let α be a type. A* basic predicate *for the type α is one of the form $(= t)$, for some $t \in \mathfrak{B}_\alpha$.*

The set $\mathbf{B}_\alpha = \{(= t) \mid t \in \mathfrak{B}_\alpha\}$ of basic predicates for the type α is called the basic language *for the type α.*

We distinguish two predicate languages that are used in belief acquisition. One is the *training predicate language* that is used in training examples. The general form of a training example for a function f is

$$\Box \forall x.((p\ x) \rightarrow (f\ x) = v),$$

where p is a predicate from the training predicate language and v is a value. It is common for training predicate languages to include the corresponding basic language (of the appropriate type).

The other language is the *hypothesis predicate language* that is used in hypotheses. The predicates appearing in a belief come from the hypothesis predicate language. In the case of learning, it would be very unlikely that the hypothesis predicate language would include any basic predicates at all (because in learning one wants to generalise beyond the training examples).

Here are two examples that illustrate some of the issues for belief acquisition.

Example 3. This example illustrates database updating which is the simplest form of belief acquisition. We show how to acquire the database of Example 1.

First, we set up the training examples. The training predicate language is the basic language $\mathbf{B}_{Occurrence}$. A typical predicate in this language is

$$(= ((21, 7, 2004), (19, 30), \mathit{WIN})).$$

The set of values is the set of basic terms $\mathfrak{B}_{Program}$. A typical value is

$$(\text{“}\mathit{Seinfeld}\text{”}, 30, [\mathit{Sitcom}], \mathit{PG}, \text{“}\mathit{Kramer}\ldots\text{”}).$$

Training examples have the form

$$\begin{aligned}
&\boldsymbol{B}_t\ \forall x.(((= ((21, 7, 2004), (19, 30), \mathit{WIN}))\ x) \longrightarrow \\
&\qquad (\mathit{tv_guide}\ x) = (\text{“}\mathit{Seinfeld}\text{”}, 30, [\mathit{Sitcom}], \mathit{PG}, \text{“}\mathit{Kramer}\ldots\text{”})) \\
&\boldsymbol{B}_t\ \forall x.(((= ((20, 7, 2004), (20, 30), \mathit{ABC}))\ x) \longrightarrow \\
&\qquad (\mathit{tv_guide}\ x) = (\text{“}\mathit{The\ Bill}\text{”}, 50, [\mathit{Drama}], \mathit{M}, \text{“}\mathit{Sun\ Hill}\ldots\text{”}))
\end{aligned}$$

and so on.

Now we choose the hypothesis predicate language. For database updating, one wants predicates in the hypothesis predicate language to pick out individuals. Thus $\mathbf{B}_{Occurrence}$ is also chosen as the hypothesis predicate language. With this choice, the belief acquisition algorithm returns the definition for the function *tv_guide* given in Example 1.

Example 4. (This example appears in [9].) 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. Suppose the current occupants are Alice, Bob, and Cathy, and that the agent knows the television preferences of each of them. Methods for acquiring these preferences were studied in [10]. Suppose that each occupant has a personal agent that has acquired (amongst many other functions) the function *likes* : $\mathit{Program} \rightarrow \Omega$, 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:

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

for suitable φ_i, ψ_i, and ξ_i. The form these can take is explained in [10].

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. (For this application, a time step could occupy hours, days, or even longer, depending on how often the beliefs need to be updated.)

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). A (simplified) predicate rewrite system suitable for giving an hypothesis predicate language for such an acquisition task is as follows.

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

Here, the function $\mathit{top} : a \rightarrow \Omega$ is defined by $(\mathit{top}\ x) = \top$, for each x. The function

$$\wedge_3 : (a \rightarrow \Omega) \rightarrow (a \rightarrow \Omega) \rightarrow (a \rightarrow \Omega) \rightarrow a \rightarrow \Omega$$

is defined by $\wedge_3\ p_1\ p_2\ p_3\ x = (p_1\ x) \wedge (p_2\ x) \wedge (p_3\ x)$, for each x. The function $\vee_2$, which defines 'disjunction' at the predicate level for two arguments, is defined analogously.

Let $\mathit{group\_likes} : \mathit{Program} \rightarrow \Omega$ be the function that the agent needs to acquire. 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.) The training predicate language is $\mathbf{B}_{\mathit{Program}}$, so that training examples for this task look like

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

where P_1 and P_2 are particular programs. The definition of a typical function that might be acquired from training examples and the hypothesis predicate 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. Suppose that a computation shows 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.

5 Reasoning with Beliefs

As well as representing knowledge, it is necessary to reason with it. The reasoning system for the logic combines a theorem prover and an equational reasoning

system. The theorem prover is a fairly conventional tableau theorem prover for modal higher-order logic similar to what is proposed in [11]. The equational reasoning system is, in effect, a computational system that significantly extends existing declarative 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. 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. However, in other applications it can just as easily be the other way around with the computational system performing subsidiary equational reasoning tasks for the theorem prover.

Here we concentrate on computation. As motivation for what computation actually means, consider the problem of determining the meaning of a term t in the intended interpretation (for some application). If a formal definition of the intended interpretation is available, then this problem can be solved (under some finiteness assumptions). However, we assume here that the intended interpretation is not available, as is usually the case, so that the problem cannot be solved directly. Nevertheless, there is still a lot that can be done if the theory $\mathcal{T}$ of the application is available and enough of it is in equational form. Intuitively, if t can be 'simplified' sufficiently using $\mathcal{T}$, its meaning may become apparent even in the absence of detailed knowledge of the intended interpretation. For example, if t can be simplified to a term containing only data constructors, then the meaning of t will generally be obvious.

More formally, the *computation problem* is as follows.

> Given a theory $\mathcal{T}$, a term t, and a sequence $\Box_{j_1} \cdots \Box_{j_r}$ of modalities, find a 'simpler' term t' such that $\Box_{j_1} \cdots \Box_{j_r} \forall (t = t')$ is a consequence of $\mathcal{T}$.

Thus t and t' have the same meaning in all worlds accessible from the point world in the intended interpretation according to the modalities $\Box_{j_1} \cdots \Box_{j_r}$.

Here now is the definition of a mechanism that addresses the computational problem by employing equational reasoning to rewrite terms to 'simpler' terms that have the same meaning. To simplify matters, we only consider the case when the computation does not need to call on the theorem prover. (This is the rank 0 case in [5].) In the following definition, a modal path to a subterm is the sequence of indices of modalities whose scope one passes through when going down to the subterm. A substitution is admissible if any term that replaces a free occurrence of a variable that is in the scope of a modality is rigid.

Definition 6. *Let $\mathcal{T} \equiv (\mathcal{G}, \mathcal{L})$ be a theory. A* computation using $\Box_{j_1} \cdots \Box_{j_r}$ with respect to $\mathcal{T}$ *is a sequence $\{t_i\}_{i=1}^{n}$ of terms such that the following conditions are satisfied.*

1. *For $i = 1, \ldots, n-1$, there is*
   (a) *a subterm s_i of t_i at occurrence o_i, where the modal path to o_i in t_i is $k_1 \ldots k_{m_i}$,*

(b) *i. a formula* $\Box_{j_1} \cdots \Box_{j_r} \Box_{k_1} \cdots \Box_{k_{m_i}} \forall(u_i = v_i)$ *in* $\mathcal{L}$, *or*
ii. a formula $\forall(u_i = v_i)$ *in* $\mathcal{G}$, *and*
(c) a substitution θ_i *that is admissible with respect to* $u_i = v_i$
such that $u_i\theta_i$ *is* α*-equivalent to* s_i *and* t_{i+1} *is* $t_i[s_i/v_i\theta_i]_{o_i}$.

The term t_1 *is called the* goal *of the computation and* t_n *is called the* answer.
Each subterm s_i *is called a* redex.
Each formula $\Box_{j_1} \cdots \Box_{j_r} \Box_{k_1} \cdots \Box_{k_{m_i}} \forall(u_i = v_i)$ *or* $\forall(u_i = v_i)$ *is called an* input equation.
The formula $\Box_{j_1} \cdots \Box_{j_r} \forall(t_1 = t_n)$ *is called the* result *of the computation.*

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. This explains the restriction to admissible substitutions in the definition of computation. It also explains why, for input equations that are local assumptions, the sequence of modalities $\Box_{k_1} \cdots \Box_{k_{m_i}}$ whose scopes are entered going down to the redex must appear in the modalities at the front of the input equation. (For input equations that are global assumptions, in effect, every sequence of modalities that we might need is implicitly at the front of the input equation.)

In the general case, an input equation can also be a theorem that was proved by the theorem-proving component of the reasoning system, as the examples below show.

Here are two examples to illustrate various aspects of computation.

Example 5. Consider a belief base for an agent that contains the definition

$$\boldsymbol{B}\ \forall x.((f\ x) = \\ \quad \textit{if } x = A \textit{ then } 42 \textit{ else if } x = B \textit{ then } 21 \textit{ else if } x = C \textit{ then } 42 \textit{ else } 0),$$

where $A, B, C : \sigma$, $f : \sigma \to \mathit{Nat}$ and $\boldsymbol{B}$ is the belief modality for the agent. With such a definition, it is straightforward to compute in the 'forward' direction. Thus $(f\ B)$ can be computed in the obvious way to produce the answer 21 and the result $\boldsymbol{B}((f\ B) = 21)$.

Less obviously, the definition can be used to compute in the 'reverse' direction. For example, consider the computation of $\{x \mid (f\ x) = 42\}$ in Figure 2, which produces the answer $\{A, C\}$. The redexes selected are underlined. This computation makes essential use of the equations

$$(w\ \ \textit{if } x \textit{ then } y \textit{ else } z) = \textit{if } x \textit{ then } (w\ y) \textit{ else } (w\ z)$$
$$(\textit{if } x \textit{ then } y \textit{ else } z\ \ w) = \textit{if } x \textit{ then } (y\ w) \textit{ else } (z\ w)$$

from the standard equality theory.

$\{x \mid \underline{(f\ x)} = 42\}$

$\{x \mid \underline{((=\ \ \mathit{if}\ x = A\ \mathit{then}\ 42\ \mathit{else}\ \mathit{if}\ x = B\ \mathit{then}\ 21\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ 42\ \mathit{else}\ 0)} 42)\}$

$\{x \mid \underline{(\mathit{if}\ x = A\ \mathit{then}\ (=\ 42)\ \mathit{else}\ (=\ \ \mathit{if}\ x = B\ \mathit{then}\ 21\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ 42\ \mathit{else}\ 0)} 42)\}$

$\{x \mid \mathit{if}\ x = A\ \mathit{then}\ \underline{(42 = 42)}\ \mathit{else}\ ((=\ \ \mathit{if}\ x = B\ \mathit{then}\ 21\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ 42\ \mathit{else}\ 0)\ 42)\}$

$\{x \mid \mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else}\ \underline{((=\ \ \mathit{if}\ x = B\ \mathit{then}\ 21\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ 42\ \mathit{else}\ 0)} 42)\}$

$\{x \mid \mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else}\ \underline{(\mathit{if}\ x = B\ \mathit{then}\ (=\ 21)\ \mathit{else}\ (=\ \ \mathit{if}\ x = C\ \mathit{then}\ 42\ \mathit{else}\ 0)} 42)\}$

$\{x \mid \mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = B\ \mathit{then}\ \underline{(21 = 42)}\ \mathit{else}\ ((=\ \mathit{if}\ x = C\ \mathit{then}\ 42\ \mathit{else}\ 0)\ 42)\}$

$\{x \mid \mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = B\ \mathit{then}\ \bot\ \mathit{else}\ \underline{((=\ \mathit{if}\ x = C\ \mathit{then}\ 42\ \mathit{else}\ 0)} 42)\}$

$\{x \mid \mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = B\ \mathit{then}\ \bot\ \mathit{else}\ \underline{(\mathit{if}\ x = C\ \mathit{then}\ (=\ 42)\ \mathit{else}\ (=\ 0)\ \ 42)}\}$

$\{x \mid \mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = B\ \mathit{then}\ \bot\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ \underline{(42 = 42)}\ \mathit{else}\ (0 = 42)\}$

$\{x \mid \mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = B\ \mathit{then}\ \bot\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ \top\ \mathit{else}\ \underline{(0 = 42)}\}$

$\{x \mid \mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = B\ \mathit{then}\ \bot\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ \top\ \mathit{else}\ \bot\}$

Fig. 2. Computation using $\boldsymbol{B}$ of $\{x \mid (f\ x) = 42\}$

Example 6. This example illustrates computation using a belief base that has been obtained by incremental belief acquisition and that exploits modalities acting on arbitrary terms. Consider an agent with belief modality $\boldsymbol{B}$ and a belief base that includes definitions of the function $f : \sigma \to \mathit{Nat}$ at the current time and some recent times. Suppose at the current time the part of the belief base concerning f is as follows.

$\boldsymbol{B}\ \forall x.((f\ x) = \mathit{if}\ (p_4\ x)\ \mathit{then}\ (\bullet f\ x)\ \mathit{else}\ \mathit{if}\ (p_5\ x)\ \mathit{then}\ 84\ \mathit{else}\ 0)$
$\bullet\boldsymbol{B}\ \forall x.((f\ x) = \mathit{if}\ (p_3\ x)\ \mathit{then}\ (\bullet f\ x)\ \mathit{else}\ 0)$
$\bullet^2\boldsymbol{B}\ \forall x.((f\ x) = \mathit{if}\ (p_1\ x)\ \mathit{then}\ 42\ \mathit{else}\ \mathit{if}\ (p_2\ x)\ \mathit{then}\ 21\ \mathit{else}\ 0)$
$\bullet^3\boldsymbol{B}\ \forall x.((f\ x) = 0).$

Three time steps ago, the function f was 0 everywhere. Two time steps ago, the definition

$\boldsymbol{B}\ \forall x.((f\ x) = \mathit{if}\ (p_1\ x)\ \mathit{then}\ 42\ \mathit{else}\ \mathit{if}\ (p_2\ x)\ \mathit{then}\ 21\ \mathit{else}\ 0)$

for f was acquired. Then, one time step ago, the definition

$\boldsymbol{B}\ \forall x.((f\ x) = \mathit{if}\ (p_3\ x)\ \mathit{then}\ (\bullet f\ x)\ \mathit{else}\ 0)$

for f was acquired. This definition states that, on the region defined by p_3, f is the same as the f at the last time step; and, otherwise, f is 0. Finally, we come to the current definition, which on the region defined by p_4 is the same as the f at the last time step; on the region defined by p_5 is 84; and, otherwise, f is 0. Definitions like these which use earlier definitions arise naturally in incremental

belief acquisition. A technical device needed to achieve incrementality is to admit values of the form $(\bullet^k f\ x)$, so that earlier definitions become available for use. In turn this depends crucially on being able to apply modalities to arbitrary terms, in this case, functions.

Now suppose t is a rigid term of type σ and consider the computation using $\boldsymbol{B}$ of $(f\ t)$ in Figure 3. Note how earlier definitions for f get used in the computation: at the step $\bullet\underline{(f\ t)}$, the definition at the last time step gets used, and at the step $\bullet^2\underline{(f\ t)}$, the definition from two time steps ago gets used.

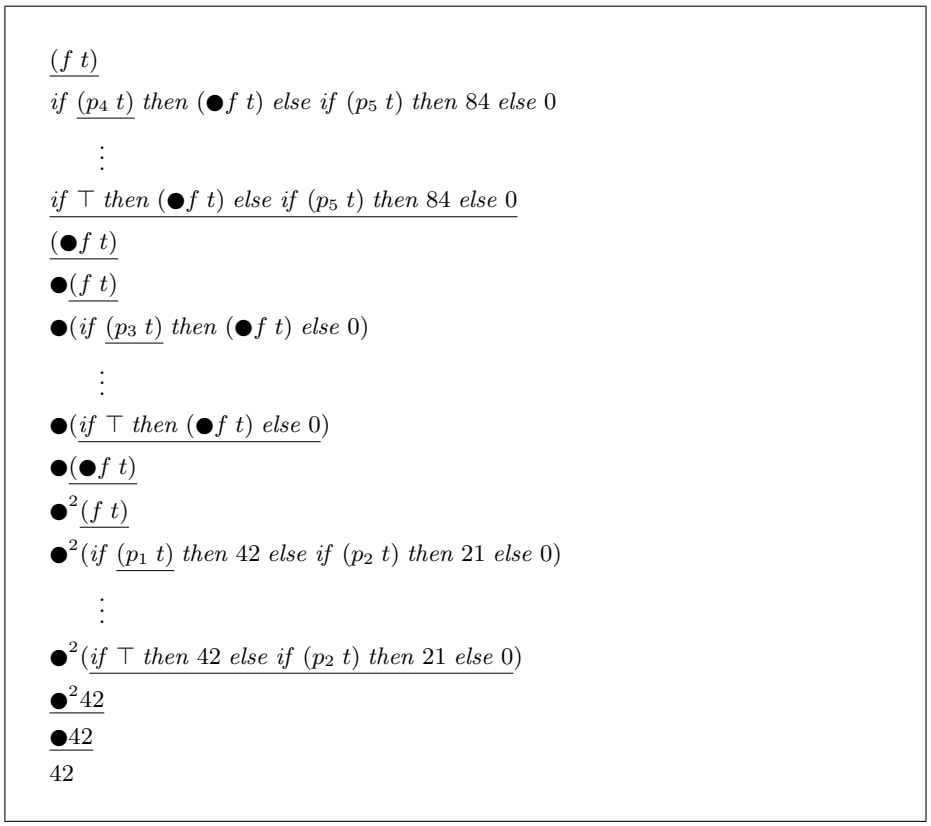

Fig. 3. Computation using $\boldsymbol{B}$ of $(f\ t)$

Also needed in this computation is the instance $(\bullet f\ t) = \bullet(f\ t)$ of the global assumption discussed in Section 2. Incidentally, the assumption that the argument to a function like f is rigid is a weak one; in typical applications, the argument will naturally be rigid.

It is assumed that the belief base of the agent contains the global assumption

$$\bullet \boldsymbol{B}\varphi \longrightarrow \boldsymbol{B}\bullet\varphi.$$

Using this assumption, it can be proved that

$$\boldsymbol{B}\bullet\, \forall x.((f\ x) = \mathit{if}\ (p_3\ x)\ \mathit{then}\ (\bullet f\ x)\ \mathit{else}\ 0)$$

and

$$\boldsymbol{B}\bullet^2\, \forall x.((f\ x) = \mathit{if}\ (p_1\ x)\ \mathit{then}\ 42\ \mathit{else}\ \mathit{if}\ (p_2\ x)\ \mathit{then}\ 21\ \mathit{else}\ 0)$$

are consequences of the belief base. These can then be used as input equations in the computation.

The computation shows that $\boldsymbol{B}((f\ t) = 42)$ is a consequence of the belief base. Thus the agent believes that the value of $(f\ t)$ is 42; on the basis of this and other similar information, it will select an appropriate action.

6 Conclusion

In this paper, we have reflected on some issues concerning beliefs for agents. The main conclusion we draw from this is the value of using a highly expressive logic for representing beliefs. Temporal and epistemic modalities allow beliefs to capture information about an environment that can be crucial when an agent is trying to select an appropriate action. For beliefs, propositional logic is not particularly useful and so it is necessary to move beyond the propositional case; we argue for the use of higher-order logic because of its extra expressive power. In spite of the expressive power of the logic (which is of course undecidable), a reasoning system in the form of a modal functional logic programming language means that agents can effectively compute using their beliefs when selecting actions. Thus reasoning during deployment of agents is substantially a programming task rather than a theorem-proving task. We are currently working on some challenging application domains for these ideas.

Acknowledgement

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

References

1. Fagin, R., Halpern, J., Moses, Y., Vardi, M.: Reasoning about Knowledge. MIT Press (1995)
2. 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)
3. Alchourrón, C., Gärdenfors, P., Makinson, D.: On the logic of theory change: Partial meet contraction and revision functions. Journal of Symbolic Logic **50**(2) (1985) 510–530
4. Mitchell, T.: Machine Learning. McGraw-Hill (1997)

5. Lloyd, J.: Knowledge representation and reasoning in modal higher-order logic. `http://users.rsise.anu.edu.au/~jwl` (2007)
6. Lloyd, J.: Logic for Learning. Cognitive Technologies. Springer (2003)
7. 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
8. Rivest, R.: Learning decision lists. Machine Learning **2**(3) (1987) 229–246
9. Lloyd, J., Ng, K.S.: Learning modal theories. In Muggleton, S., Otero, R., Tamaddoni-Nezhad, A., eds.: Proceedings of the 16th International Conference on Inductive Logic Programming (ILP 2006), Springer, LNAI 4455 (2007) 320–334
10. Cole, J., Gray, M., Lloyd, J., Ng, K.S.: Personalisation for user agents. In Dignum, F., *et al*, eds.: Fourth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 05). (2005) 603–610
11. Fitting, M.: Types, Tableaus, and Gödel's God. Kluwer Academic Publishers (2002)