An Architecture for Rational Agents

J.W. Lloyd and T.D. Sears

Computer Sciences Laboratory
Research School of Information Sciences and Engineering
The Australian National University
{jwl,timsears}@csl.anu.edu.au

Abstract. This paper is concerned with designing architectures for rational agents. In the proposed architecture, agents have belief bases that are theories in a multi-modal, higher-order logic. Belief bases can be modified by a belief acquisition algorithm that includes both symbolic, on-line learning and conventional knowledge base update as special cases. A method of partitioning the state space of the agent in two different ways leads to a Bayesian network and associated influence diagram for selecting actions. The resulting agent architecture exhibits a tight integration between logic, probability, and learning. Two illustrations of the agent architecture are provided, including a user agent that is able to personalise its behaviour according to the user's interests and preferences.

1 Introduction

Our starting point is the conventional view that an agent is a system that takes percept sequences as input and outputs actions [9, p.33]. Naturally enough, on its own, this barely constrains the agent architectures that would be feasible. Thus we add a further constraint that agents should act rationally, where a rational agent is defined as follows [9, p.36]:

> "For each possible percept sequence, a rational agent should select an action that is expected to maximise its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has".

Various specific rationality principles could be used; we adopt the well-known principle of maximum expected utility [9, p.585] (namely, a rational agent should choose an action that maximises the agent's expected utility). This implies that, for the intended applications, it is possible for the agent designer to specify utilities for all states (or, at least, the agent has some means of acquiring these utilities). The proposed architecture includes a decision-theoretic component involving utilities and Bayesian networks that implements the principle of maximum expected utility.

Another major component of the architecture is a model of the environment (or, at least, a model of enough of the environment in order to be able to effectively select actions). This model has two parts: state and beliefs. The state

records some information that helps the agent mitigate the well-known problems that arise from partial observability. The beliefs are random variables defined on the state; evidence variables assist in the selection of actions and result variables assist in the evaluation of the utility of states. Beliefs are expressed in a multi-modal, higher-order logic.

Implicit in the concept of rationality is the notion that agents should make every effort to acquire whatever information is needed for action selection. This implies that agents should have a learning component that allows them to improve their performance (which includes adapting to changing environments). The proposed architecture incorporates a learning component for exactly this purpose.

The resulting agent architecture exhibits a tight integration between logic, probability, and learning. In the terminology of [9, p.51], agents employing this architecture are model-based, utility-based, learning agents.

An outline of this paper is as follows. The next section provides a brief introduction to beliefs and their acquisition. Section 3 describes the approach to agent architecture. Section 4 gives two illustrations of the agent architecture. The paper concludes with some remarks about the wider context of this research.

2 Beliefs

This section contains a brief introduction to belief bases and the logic in which these are expressed. We employ the logic from [5] which is a multi-modal version of the higher-order logic in [4] (but leaving out polymorphism); much more detail about the logic is contained in these two works.

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 constructors *1* and Ω both of arity 0. *1* is the type of some distinguished singleton set and 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 $x, y, z, \ldots$. For any particular application, the alphabet is assumed fixed and all definitions are relative to the alphabet.

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. (For* $k = 0$*, this reduces to a type constructor of arity* 0 *being a type.)*

2. *If α and β are types, then $\alpha \to \beta$ is a type.*
3. *If $\alpha_1, \ldots, \alpha_n$ are types, then $\alpha_1 \times \cdots \times \alpha_n$ is a type. (For $n = 0$, this reduces to 1 being a type.)*

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

1. (), having signature $\mathit{1}$.
2. $=_\alpha$, having signature $\alpha \to \alpha \to \Omega$, for each type α.
3. $\top$ and $\bot$, having signature Ω.
4. $\neg$, having signature $\Omega \to \Omega$.
5. $\wedge$, $\vee$, $\longrightarrow$, $\longleftarrow$, and $\longleftrightarrow$, having signature $\Omega \to \Omega \to \Omega$.
6. 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 $\top$.

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 $\alpha_1 \times \cdots \times \alpha_n$ (for $n \geq 0$).*
6. *If t is a term of type Ω and $i \in \{1, \ldots, m\}$, then $\Box_i t$ is a term of type Ω.*

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). A formula of the form $\Box_i \varphi$ is interpreted as 'agent i believes φ'.

An important feature of higher-order logic is that it admits functions that can take other functions as arguments and thus has greater expressive power for knowledge representation than first-order logic. This fact is exploited throughout the architecture, in the use of predicates to represent sets and in the predicate rewrite systems used for learning, for example.

In applications there are typically many individuals that need to be represented. For example, with agent applications, the state is one such individual. With logic as the representation language, (closed) terms represent individuals. In [4], a class of terms, called basic terms, is identified for this purpose. The inductive definition of basic terms comes in three parts. The first part uses data constructors to represent numbers, lists, and so on. The second part uses abstractions to represent sets, multisets, and so on. The third part uses tuples to represent vectors. The class of basic terms provides a rich class of terms for

representing a variety of structured individuals. We use basic terms to represent individuals in the following.

As shown in [4], and also [5], a functional logic programming computational model can be used to evaluate functions that have equational definitions in the logic. Furthermore, [5] provides a tableau theorem-proving system for the multi-modal, higher-order logic.

A method of belief acquisition for belief bases that are theories in the logic is under development. The belief acquisition algorithm includes both symbolic, on-line learning and conventional knowledge base update as special cases. The key idea of the algorithm is to introduce two languages, the training language and the hypothesis language. By carefully controlling these lanaguges it is possible to have belief acquisition that at one extreme directly incorporates new beliefs into the belief bases (as for a conventional knowledge base update algorithm) and at the other extreme is a conventional learning algorithm that learns definitions of functions that generalise to new individuals. The algorithm itself is a (somewhat generalised) decision-list learning algorithm, as introduced in [8]. Thus definitions of functions in belief bases are (logical forms of) decision lists.

The basic idea for the use of the logic is that each agent in a multi-agent system has its own belief base that consists of formulas of the form $\Box_i \varphi$ (for agent i). Also agents can have access to the belief bases of other agents by means of interaction axioms which are schemas of the form

$$\Box_i \varphi \longrightarrow \Box_j \varphi,$$

whose intuitive meaning is that 'if agent i believes φ, then agent j believes φ'. In addition, while not illustrated in this paper, standard modal axioms such as T, D, B, 4, and 5 can be used for any modality $\Box_i$, depending on the precise nature of the beliefs in the belief base of agent i. The tableau theorem-proving system in [5] can prove theorems that involve these standard modal axioms and interaction axioms.

Finally, note that while one could use the logic for *specification* of agents (as in [10], for example), this is not what is proposed here. Instead, we use the logic for expressing actual belief bases of agents, and theorem proving and computation involving these belief bases is an essential component of the *implementation* of agents.

3 Agent Architecture

In this section, we describe the agent architecture.

We consider an agent situated in some environment that can receive percepts from the environment and can apply actions to the environment. Thus the agent can be considered to be a function from percepts to actions; an agent architecture provides the definition of this function.

Let S denote the set of states of the agent. Included in a state may be information about the environment or something that is internal to the agent. It is also likely to include the agent's current intention or goal, that is, what it

is currently trying to achieve. The state may be updated as a result of receiving a percept. For example, a user agent may change its intention due to a request from the user.

As well as some state, the agent's model includes its belief base, which can also be updated. In the proposed architecture, belief bases have a particular form that we introduce below motivated by the desire to make action selection as effective as possible.

Let A denote the set of actions of the agent. Each action changes the current state to a new state (possibly non-deterministically). The agent selects an action that maximises the expected utility. Executing the action involves applying the action to the environment and/or moving to a new state.

Summarising the description so far, here is a high-level view of the main loop of the agent algorithm.

`loop forever`
  get percept
  update model
  select action
  put action.

We now concentrate on the action selection part of this loop. In practical applications, the set of states may be very large; in this case, it may be impractical to try to select the action by explicitly dealing with all these states. An obvious idea is to partition the state space so that the behaviour of an action is somehow consistent over all the states in an equivalence class [1]. We exploit this idea in what follows.

For that we will need some random variables. It will be helpful to be quite precise about their definition. Random variables are functions; this fact and the probability space that they are defined on will be important in the description of the agent architecture. For simplicity and because it is the case of most interest in applications, we confine the discussion to random variables that are discrete. (Note that $(f \circ g)(x)$ means $g(f(x))$.)

Definition 4. *A* random variable *is a measurable function $X : \Omega \to \Xi$, where $(\Omega, \mathcal{A}, p)$ is a probability space and $(\Xi, \mathcal{B})$ is a measurable space. The probability measure $X^{-1} \circ p$ on $\mathcal{B}$ is called the* distribution *(or* law*) of X.*

The probability space of interest here is the product space $S \times A \times S$ which is assumed to have some suitable σ-algebra and probability measure $p(\cdot, \cdot, \cdot)$ defined on it. Intuitively, $p(s, a, s')$ is the probability that applying action a will result in a transition from state s to state s'. By conditioning on states and actions, for each state $s \in S$ and action $a \in A$, a transition probability distribution $p(\cdot \mid s, a)$ is obtained.

There are three projections defined on $S \times A \times S$. The first projection *initial* : $S \times A \times S \to S$ is defined by

$$\mathit{initial}(s, a, s') = s,$$

the second projection $action : S \times A \times S \to A$ is defined by

$$action(s, a, s') = a,$$

and the third projection $final : S \times A \times S \to S$ is defined by

$$final(s, a, s') = s',$$

for each $(s, a, s') \in S \times A \times S$. Each projection induces a probability measure on their respective codomains that makes the codomains into probability spaces.

We will be interested in two different sets of random variables on S. These are the *evidence variables* $E_i : S \to V_i$ $(i = 1, \ldots, n)$ and the *result variables* $R_j : S \to W_j$ $(j = 1, \ldots, m)$. Two sets of random variables can now be defined on the probability space $S \times A \times S$, as follows. For each evidence variable E_i, consider the random variable $initial \circ E_i : S \times A \times S \to V_i$. Similarly, for each result variable R_j, consider the random variable $final \circ R_j : S \times A \times S \to W_j$. Let X denote $(initial \circ E_1, \ldots, initial \circ E_n, action, final \circ R_1, \ldots, final \circ R_m)$. All this can now be put together to obtain the random variable

$$X : S \times A \times S \to V_1 \times \cdots \times V_n \times A \times W_1 \times \cdots \times W_m,$$

which is illustrated in Figure 1. The distribution of X is given by $X^{-1} \circ p$, where p is the probability measure on $S \times A \times S$.

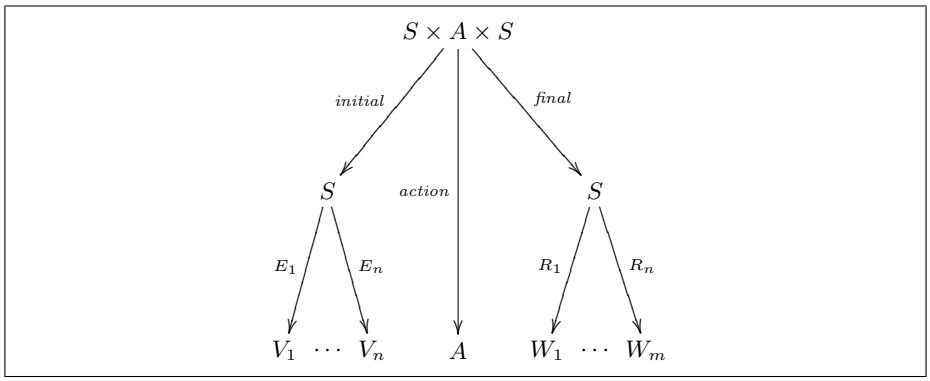

Fig. 1. Evidence and result variables

In the following, we will refer to an element of $V_1 \times \cdots \times V_n$ as an *evidence tuple* and an element of $W_1 \times \cdots \times W_m$ as a *result tuple*.

The distribution of X is illustrated in the influence diagram in Figure 2. Here the Bayesian network given by the evidence and result variables is extended into an influence diagram by adding the random variable *action* for the action selected and a utility node to indicate that it is the result variables that contribute to the utility. Each node in the Bayesian network has an associated (conditional)

probability table (that is not shown in Figure 2). Note that, in general, it is possible for there to be dependencies amongst the $initial \circ E_i$ and amongst the $final \circ R_j$.

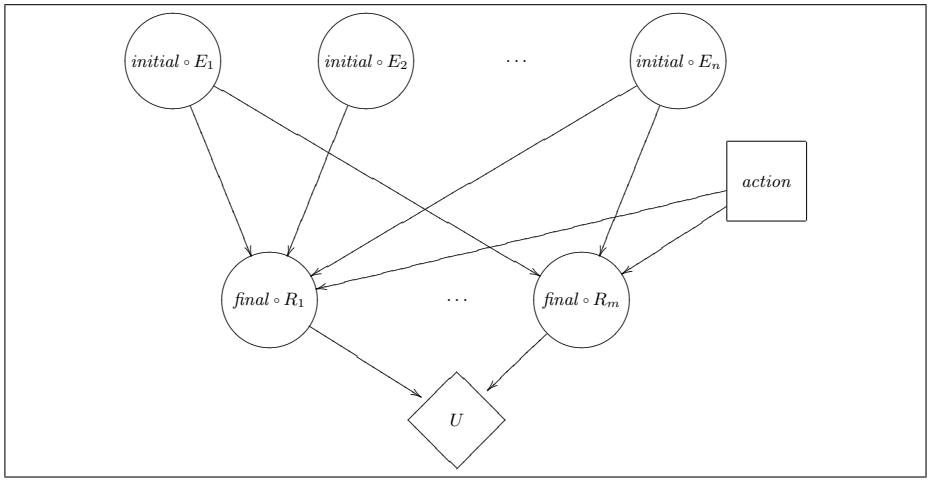

Fig. 2. Influence diagram

The evidence and result variables play an important role in action selection. One can think of these as features of the states. However, each class of features serves a different purpose. The *evidence variables are chosen so as to assist the selection of a good action*, whereas the *result variables are chosen so as to provide a good evaluation of the resulting state*. It will be convenient to insist that included amongst the evidence variables are boolean random variables that describe the pre-condition (if any) of each action.

These random variables will be functions with definitions in the belief base. We can now be more precise about the definition of belief bases.

Definition 5. *A* belief base *is a theory consisting of the definitions of the evidence variables and (some of) the result variables.*

The implication of this definition is that the belief base should contain the definitions of the evidence variables, (some of) the result variables, and any subsidiary functions used to define these, and that is *all* it need contain – there is no other use for functions in the belief base. This fact provides strong guidance during the design of the agent. Observations of the result variables will be needed but it will not be essential for the agent to know their (full) definitions. An application for which the definition of a result variable is not known but for which the variable can be adequately observed is given below.

At this point in the development we have entered the realm of graphical probabilistic models. Over the last 20 years or so there has been a huge effort

to find efficient algorithms for doing all kinds of tasks such as marginalising, conditioning, and learning in graphical probabilistic models with large numbers of random variables. (See, for example, [3] and [6].) All of this work is directly relevant to the agent context and we hope to exploit it in future. For the moment, we are primarily interested in decision making which does not require the full graphical model and the applications of current interest have small graphical models anyway, so we simply make a few remarks below about this context. On the other hand, a new problem is raised by the agent applications in that the definitions of the random variables are generally changing and this leads to some complications that deserve a deeper investigation.

By conditioning on the evidence variables and action variable in Figure 2, the table in Figure 3 is obtained. There is a row in this table for every combination of an evidence tuple together with an action, except that rows for which the precondition (if any) of the action has value $\perp$ are deleted. There is a column under $\mathit{final} \circ (R_1, \ldots, R_m)$ for every result tuple. For each combination of evidence tuple and action, there is an associated transition probability distribution that gives, for each possible result tuple, the probability of reaching that result tuple. The last row records the utilities for each result tuple. To compute the expected utility $EU(e, a)$ of action a applied to evidence tuple e, we proceed as follows. Suppose that e and a appear together in the ith row of the table. Then the expected utility is

$$EU(e, a) = \sum_{j=1}^{l_1 \cdots l_m} p_{i,j} \times u_j.$$

		$\mathit{final} \circ (R_1, \ldots, R_m)$			
$\mathit{initial} \circ (E_1, \ldots, E_n)$	action	$(w_{1,1}, \ldots, w_{m,1})$	$(w_{1,2}, \ldots, w_{m,1})$	$\ldots$	$(w_{1,l_1}, \ldots, w_{m,l_m})$
$(v_{1,1}, \ldots v_{1,n})$	a_1	$p_{1,1}$	$p_{1,2}$	$\ldots$	$p_{1,l_1 \cdots l_m}$
$(v_{2,1}, \ldots v_{2,n})$	a_2	$p_{2,1}$	$p_{2,2}$	$\ldots$	$p_{2,l_1 \cdots l_m}$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
$(v_{k,1}, \ldots v_{k,n})$	a_k	$p_{k,1}$	$p_{k,2}$	$\ldots$	$p_{k,l_1 \cdots l_m}$
		u_1	u_2	$\ldots$	$u_{l_1 \cdots l_m}$

Fig. 3. Influence diagram conditioned on the evidence and action variables

A policy for an agent is a mapping $\mathit{policy} : S \to A$. The policy is extracted from the conditioned influence diagram in the usual way. Given a state s, the action a selected by *policy* is the one for which $EU((E_1(s), \ldots, E_n(s)), a)$ is a maximum. The case when the state s is not known exactly but is given by a distribution (the 'belief state' case [9]) is handled by the obvious generalisation of this. This policy thus implements the principle of maximum expected utility.

The agent architecture exhibits a tight integration between logic and probability, since the random variables in the network have definitions given by the

logic. Furthermore, for agents in dynamic situations, the definitions of evidence variables will need to be modified from time to time. In the first illustration of the next section, we discuss a personalisation application for which there are three evidence variables, one of which is learned and the other two are subject to updating. In effect, the evidence variables are tailored to the interests and preferences of the user and this will be seen to be essential for selecting good actions.

The approach taken here to agent architecture assumes that it is possible to specify the utilities. Generally, assigning utilities to a very large number of states in advance is a difficult task. However, with the right choice of result variables, the task can become much easier; this is illustrated with the applications in the next section. The modelling challenge for the agent designer is to find result variables so that all states that map to the same result tuple really do have the same (or very similar) utility. Thus, in essence, the task is to find a good piecewise constant approximation of the utility function. While not every application will succumb to this approach, it seems that there is a sufficiently large subset of agent applications which does and therefore the approach is useful.

Having settled on the actions, and evidence and result variables, and having specified the utilities, the only other information needed to obtain a policy is the collection of transition probability distributions $p(\cdot \mid e, a)$ in Figure 3. How are these obtained? Essentially all that needs to be done is to observe triples of the form (e, a, r), where e is an evidence tuple, a is an action, and r is the corresponding result tuple. The observation of each such triple increments a count kept at the corresponding entry in the table of transition probabilities. More generally, r is not uniquely determined by e and a but is given by a probability distribution. In this case, the increment is proportioned over the result tuples according to this distribution. These counts can then be normalised over each row to obtain a probability distribution.

4 Illustrations

We now consider two illustrations of the agent architecture.

4.1 TV Recommender

The first illustration is concerned with applying machine learning techniques to building user agents that facilitate interaction between a user and the Internet. It concentrates on the topic of personalisation in which the agent adapts its behaviour according to the interests and preferences of the user. There are many practical applications of personalisation that could exploit this technology.

This illustration is set in the context of an infotainment agent, which is a multi-agent system that contains a number of agents with functionalities for recommending movies, TV programs, music and the like, as well as information agents with functionalities for searching for information on the Internet. Here we

concentrate on the TV recommender as a typical such agent. More detail (but not the decision-theoretic discussion that follows) is given in [2].

A detailed description of the most pertinent aspects of the design of the TV recommender is now given. The knowledge representation aspects of the TV recommender are presented first; these mainly concern the states, actions, evidence variables, and result variables.

First we introduce the types that will be needed and the data constructors corresponding to these types. We will need several standard types: Ω (the type of the booleans), *Nat* (the type of natural numbers), *Int* (the type of integers), and *String* (the type of strings). Also *List* denotes the (unary) list type constructor. Thus, if α is a type, then *List* α is the type of lists whose elements have type α.

We introduce the following type synonyms.

$$State = Occurrence \times Status$$
$$Occurrence = Date \times Time \times Channel$$
$$Date = Day \times Month \times Year$$
$$Time = Hour \times Minute$$
$$Program = Title \times Subtitle \times Duration \times (List\ Genre) \times Classification \times Synopsis$$
$$Text = List\ String.$$

In addition, *Title*, *Subtitle*, and *Synopsis* are all defined to be *String*, and *Year*, *Month*, *Day*, *Hour*, *Minute* and *Duration* are all defined to be *Nat*.

The data constructors for the type *Status* are as follows.

$$Unknown, Yes, No : Status.$$

The meaning of *Unknown* is that a recommendation (about a program having a particular occurrence) hasn't yet been made, *Yes* means that it has a positive recommendation, and *No* means that it has a negative recommendation.

Channel is the type of TV channels, *Genre* the type of genres, and *Classification* the type of classifications. There are 49 channels, 115 genres and 7 classifications.

There is a type *Action* with data constructors given by

$$RecommendYes, RecommendNo : Action.$$

The action *RecommendYes* is a positive recommendation, while *RecommendNo* is a negative recommendation. The action *RecommendYes* takes a state $(o, Unknown)$, where o is some occurrence, and produces the new state (o, Yes). Similarly, the action *RecommendNo* takes a state $(o, Unknown)$ and produces the new state (o, No).

In the following, the definitions of various functions will appear. These are (mainly) in the belief base of the TV agent. To indicate that they are beliefs of the TV agent, the necessity modality $\Box_t$ is used. Thus, if φ is a formula, then the meaning of $\Box_t\varphi$ is that 'φ is a belief of the TV agent'. Other agents in the multi-agent system have their own necessity modality; for example, the modality for the diary agent is $\Box_d$. There is also a base of common beliefs accessible to

all the agents for which the necessity modality is $\Box$. Interaction axioms allow one agent to access the beliefs of another agent [5]. So, for example, there are interaction axioms that allow the TV agent to access the beliefs of the diary agent and also the common beliefs.

There are three projections on $State \times Action \times State$. The first projection is

$$initial : State \times Action \times State \rightarrow State$$
$$\Box_t \, \forall_{State} s. \, \forall_{Action} a. \, \forall_{State} s'.$$
$$((initial \ (s, a, s')) =_{State} s).$$

The second projection is

$$action : State \times Action \times State \rightarrow Action$$
$$\Box_t \, \forall_{State} s. \, \forall_{Action} a. \, \forall_{State} s'.$$
$$((action \ (s, a, s')) =_{Action} a).$$

The third projection is

$$final : State \times Action \times State \rightarrow State$$
$$\Box_t \, \forall_{State} s. \, \forall_{Action} a. \, \forall_{State} s'.$$
$$((final \ (s, a, s')) =_{State} s').$$

Now we turn to the evidence variables. To define these, a number of subsidiary functions are needed and so we start there.

The agent has access via the Internet to a TV guide (for the next week or so) for all channels. This database is represented by a function *tv_guide* having signature

$$tv\_guide : Occurrence \rightarrow Program.$$

Here the date, time and channel information uniquely identifies the program and the value of the function is (information about) the program itself. The TV guide consists of (thousands of) facts like the following one.

$$\Box_t \ ((tv\_guide \ ((20, 7, 2004), (20, 30), ABC)) =_{Program}$$
$$(\text{“}The\ Bill\text{”}, \text{“”}, 50, [Drama], M,$$
$$\text{“}Sun\ Hill\ continues\ to\ work\ at\ breaking\ the\ people\ smuggling\ operation\text{”})).$$

This fact states that the program on 20 July 2004 at 8.30pm on channel ABC has title “The Bill”, no subtitle, a duration of 50 minutes, genre drama, a classification for mature audiences, and synopsis “Sun Hill continues to work at breaking the people smuggling operation”.

There are a number of simple subsidiary functions that are defined as follows. Note that the definition of the function *add* is in the base of common beliefs and hence uses the modality $\Box$.

$$proj_{Occurrence} : State \rightarrow Occurrence$$
$$\Box_t \, \forall_{Occurrence} o. \, \forall_{Status} s.$$
$$((proj_{Occurrence} \ (o, s)) =_{Occurrence} o).$$

$$
\begin{array}{l}
period : Occurrence \rightarrow Date \times Time \times Time \\
\Box_t \; \forall_{Date} d.\; \forall_{Time} t.\; \forall_{Channel} c. \\
\quad ((period\ (d,t,c)) =_{Date \times Time \times Time} \\
\quad\quad (d,t,(add\ (t,(proj_{Duration}\ (tv\_guide\ (d,t,c))))))).
\end{array}
$$

$$
\begin{array}{l}
add : Time \times Duration \rightarrow Time \\
\Box\ \forall_{Hour} h.\; \forall_{Minute} m.\; \forall_{Duration} d. \\
\quad ((add\ ((h,m),d)) =_{Time} \\
\quad\quad ((60 \times h + m + d)\ div\ 60, (60 \times h + m + d)\ mod\ 60)).
\end{array}
$$

The belief base of the TV recommender includes the function *user_tv_time_acceptable* which has a definition that is obtained by belief acquisition and would look something like the following.

$$
\begin{array}{l}
user\_tv\_time\_acceptable : Date \times Time \times Time \rightarrow \Omega \\
\Box_t \; \forall_{Date} d.\; \forall_{Time} t.\; \forall_{Time} t'. \\
\quad ((user\_tv\_time\_acceptable\ (d,t,t')) =_{\Omega} \\
\quad\quad if\ (weekday\ d) \wedge ((proj_{Hour}\ t) \geq 20) \wedge ((proj_{Hour}\ t') \leq 23)\ then\ \top \\
\quad\quad else\ if\ \neg(weekday\ d) \wedge ((proj_{Hour}\ t) \geq 12) \wedge ((proj_{Hour}\ t') \leq 25)\ then\ \top \\
\quad\quad else\ \bot).
\end{array}
$$

This definition states that the user is willing to watch programs that start between 8pm and 11pm on weekdays and between midday and 1am on weekends. This information is obtained rather directly from the user by the belief acquisition algorithm.

The belief base of the TV recommender also includes the function *user_likes_tv_program* which has a definition that is obtained by belief acquisition (in this case, machine learning since the function learned generalises) and would look something like the following.

$$
\begin{array}{l}
user\_likes\_tv\_program : Program \rightarrow \Omega \\
\Box_t \; \forall_{Program} x. \\
\quad ((user\_likes\_tv\_program\ x) =_{\Omega} \\
\quad\quad if\ (proj_{Title} \circ (=_{Title}\ \text{“}NFL\ Football\text{”})\ x)\ then\ \top \\
\quad\quad else\ if\ (proj_{(List\ Genre)} \circ (listExists_1\ genre \circ (<0))\ x)\ then\ \bot \\
\quad\quad else\ if\ (proj_{Title} \circ StringToText \circ (listExists_1\ (=_{String}\ \text{“}sport\text{”}))\ x)\ then\ \top \\
\quad\quad else\ if\ (proj_{(List\ Genre)} \circ (listExists_1\ (=_{Genre}\ Current\_Affairs))\ x)\ then\ \bot \\
\quad\quad else\ if\ (proj_{Synopsis} \circ StringToText \circ (listExists_1\ (=_{String}\ \text{“}american\text{”}))\ x)\ then\ \top \\
\quad\quad\quad\quad \vdots \\
\quad\quad else\ \bot).
\end{array}
$$

Much more detail on how this function is learned is contained in [2].

Another subsidiary function needed is *user_diary_free* that has signature

$$user\_diary\_free : Date \times Time \times Time \rightarrow \Omega.$$

The definition of this function is in the belief base of the diary agent. The TV agent has access to this definition because of an interaction axiom that makes the belief base of the diary agent accessible to the TV agent.

We can now define the evidence variables.

The first evidence variable is

$$\begin{array}{l} ultp : State \rightarrow \Omega \\ \Box_t \, \forall_{State} x. \\ \quad ((ultp\ x) =_\Omega \\ \quad\quad (proj_{Occurrence} \circ tv\_guide \circ user\_likes\_tv\_program\ x)). \end{array}$$

This evidence variable is intended to indicate whether or not the user likes a program (irrespective of whether or not the user would be able to watch it).

The second evidence variable is

$$\begin{array}{l} udiary : State \rightarrow \Omega \\ \Box_t \, \forall_{State} x. \\ \quad ((udiary\ x) =_\Omega \\ \quad\quad (proj_{Occurrence} \circ period \circ user\_diary\_free\ x)). \end{array}$$

This evidence variable is intended to indicate whether or not the user's diary is free during the time that the program is on.

The third evidence variable is

$$\begin{array}{l} uaccept : State \rightarrow \Omega \\ \Box_t \, \forall_{State} x. \\ \quad ((uaccept\ x) =_\Omega \\ \quad\quad (proj_{Occurrence} \circ period \circ user\_tv\_time\_acceptable\ x)). \end{array}$$

This evidence variable is intended to indicate whether or not the user is willing to watch television at the time the program is on.

The intuition is that the three evidence variables are features that together can reasonably be used by the TV agent to make recommendations for the user.

Next we turn to the result variables.

The first result variable is the function

$$user : State \rightarrow \Omega$$

which models the user as an oracle about occurrences of programs. Given a state, *user* returns $\top$ if the user intends to watch the program whose occurrence is in the first component of the state; otherwise, *user* returns $\bot$. Of course, the

definition of *user* is not available to the agent. But it can observe values for this function by asking the user.

The second result variable is

$$
\begin{array}{l}
\mathit{recomm} : \mathit{State} \rightarrow \mathit{Status} \\
\Box_t \, \forall_{\mathit{Occurrence}} o. \, \forall_{\mathit{Status}} s. \\
\quad ((\mathit{recomm}\ (o, s)) =_{\mathit{Status}}\ s).
\end{array}
$$

The function *recomm* simply projects onto the status component of the state.

Figure 4 illustrates the evidence and result variables for the TV recommender. The corresponding influence diagram is given in Figure 5.

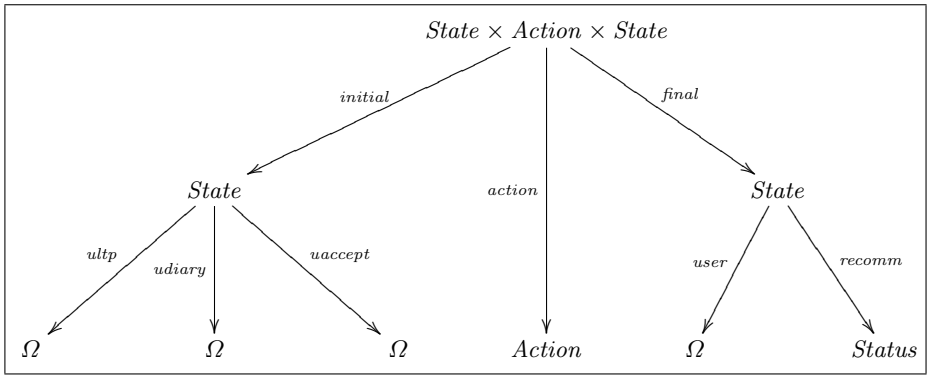

Fig. 4. Evidence and result variables for the TV recommender

Given in Figure 6 is a set of transition probability distributions and utilities that could reasonably arise in practice. (The utilities are given by the user.) The last column contains the expected utility of the action given the evidence for each row. In Figure 6 it is the case that the definition acquired by the agent for *uaccept* is actually not all that accurate and the user is willing to watch programs outside the periods given in the definition of this function. Secondly, it is assumed that the user is reasonably happy with states in which the agent recommends programs that they do not actually want to watch and therefore gives this state the utility value of 0.4. The policy corresponding to the situation is given in Figure 7 and its definition in the logic is as follows.

$$
\begin{array}{l}
\mathit{policy} : \mathit{State} \rightarrow \mathit{Action} \\
\Box_t \, \forall_{\mathit{State}} s. \\
\quad ((\mathit{policy}\ s) =_{\mathit{Action}} \\
\quad\quad \mathit{if}\ (((\mathit{ultp}\ s) =_{\Omega} \top) \wedge ((\mathit{udiary}\ s) =_{\Omega} \top))\ \mathit{then}\ \mathit{RecommendYes} \\
\quad\quad \mathit{else}\ \mathit{RecommendNo}).
\end{array}
$$

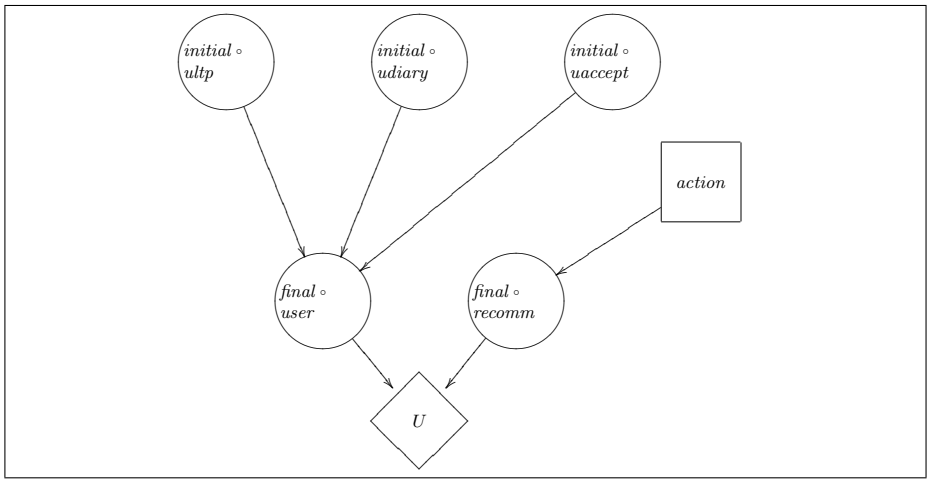

Fig. 5. Influence diagram for the TV recommender

		$final \circ (user, recomm)$				
$initial \circ (ultp, udiary, uaccept)$	$action$	$(\top, Yes)$	$(\top, No)$	$(\bot, Yes)$	$(\bot, No)$	
$(\top, \top, \top)$	$RecommendYes$	0.8	0.0	0.2	0.0	0.88
$(\top, \top, \top)$	$RecommendNo$	0.0	0.8	0.0	0.2	0.20
$(\top, \top, \bot)$	$RecommendYes$	0.4	0.0	0.6	0.0	0.64
$(\top, \top, \bot)$	$RecommendNo$	0.0	0.4	0.0	0.6	0.60
$(\top, \bot, \top)$	$RecommendYes$	0.0	0.0	1.0	0.0	0.40
$(\top, \bot, \top)$	$RecommendNo$	0.0	0.0	0.0	1.0	1.00
$(\top, \bot, \bot)$	$RecommendYes$	0.0	0.0	1.0	0.0	0.40
$(\top, \bot, \bot)$	$RecommendNo$	0.0	0.0	0.0	1.0	1.00
$(\bot, \top, \top)$	$RecommendYes$	0.1	0.0	0.9	0.0	0.46
$(\bot, \top, \top)$	$RecommendNo$	0.0	0.1	0.0	0.9	0.90
$(\bot, \top, \bot)$	$RecommendYes$	0.0	0.0	1.0	0.0	0.40
$(\bot, \top, \bot)$	$RecommendNo$	0.0	0.0	0.0	1.0	1.00
$(\bot, \bot, \top)$	$RecommendYes$	0.0	0.0	1.0	0.0	0.40
$(\bot, \bot, \top)$	$RecommendNo$	0.0	0.0	0.0	1.0	1.00
$(\bot, \bot, \bot)$	$RecommendYes$	0.0	0.0	1.0	0.0	0.40
$(\bot, \bot, \bot)$	$RecommendNo$	0.0	0.0	0.0	1.0	1.00
		1	0	0.4	1	

Fig. 6. Influence diagram conditioned on the evidence and action variables for the TV recommender

$initial \circ (ultp, udiary, uaccept)$	$action$
$(\top, \top, \top)$	$RecommendYes$
$(\top, \top, \bot)$	$RecommendYes$
$(\top, \bot, \top)$	$RecommendNo$
$(\top, \bot, \bot)$	$RecommendNo$
$(\bot, \top, \top)$	$RecommendNo$
$(\bot, \top, \bot)$	$RecommendNo$
$(\bot, \bot, \top)$	$RecommendNo$
$(\bot, \bot, \bot)$	$RecommendNo$

Fig. 7. Policy for the TV recommender corresponding to Figure 6

4.2 Blocks World

The second illustration is the traditional blocks world domain, which will highlight a couple of aspects missing from the TV recommender.

Here are some declarations suitable for this domain.

$$B_0, B_1, B_2, B_3, B_4, B_5, B_6, B_7, B_8, B_9, Floor : Object$$
$$Stack = List\ Object$$
$$World = \{Stack\}$$
$$OnState = Object \times Object$$
$$Intention = \{OnState\}$$
$$State = World \times Intention.$$

We consider worlds in which there are up to 10 blocks. A blocks world is modelled as a set of stacks of blocks. The agent's intentions are modelled as a set of on-states, where an *on-state* is a pair of objects, the first of which is intended to be immediately on top of the second. The set of on-states may not fully specify the position of all blocks in the world; an intention is just a set of constraints that any goal state should satisfy. A state is a pair consisting of a blocks world and an intention.

A block b in the world of a state is *misplaced* if (i) the pair consisting of b and its supporting object directly contradicts one of the on-states in the intention, or (ii) b is supported by a misplaced block. A move of a block is *constructive* if, after the move, the block is not misplaced and the move achieves an on-state in the intention. Once a block has been moved constructively, it need not move again in the course of achieving the intention.

This discussion leads to the definition of two actions. The first, called *CMove*, makes a constructive move of a block. If there are several blocks that can be moved constructively, then the choice of block to be moved is non-deterministic. The second action, called *FMove*, involves moving a misplaced block (that is not on the floor) to the floor. Once again the choice of such a block to be moved is non-deterministic. This gives the following declaration of the constants for the

type *Action*.

$CMove, FMove : Action.$

It is assumed that after executing an action, the agent receives a percept from the environment that enables it to know the exact state of the world (that is, which particular block was actually moved).

Next we give the evidence variables which have the following signatures.

$okC : State \to \Omega$
$okF : State \to \Omega.$

The intended meaning of predicate *okC* is that it is true iff a constructive move is possible. The intended meaning of predicate *okF* is that it is true iff there is a misplaced block that is not on the floor. Note that, if a world does not satisfy the intention and *okC* is false, then *okF* is true.

The precondition for the action *CMove* is that *okC* is true. Also the precondition for the action *FMove* is that *okF* is true. Thus, for blocks world, the evidence variables simply provide the preconditions for the actions.

There is a single result variable *misplaced* that returns the number of blocks in the world that are misplaced (with respect to the intention). The function *misplaced* has a definition as follows.

$misplaced : State \to Nat$
$\Box_b \; \forall_{State} s.$
$\quad ((misplaced\ s) =_{Nat}$
$\quad\quad card\ \{b \mid (isMisplaced\ b\ s)\}).$

Here $\Box_b$ is the necessity modality for the blocks world agent, *card* computes the cardinality of a set, and *isMisplaced* is a predicate that checks whether a block is misplaced for a state. Intuitively, states for which the value of *misplaced* is smaller should have a higher utility, as they are 'closer' to a goal state.

The evidence and result variables are shown in Figure 8.

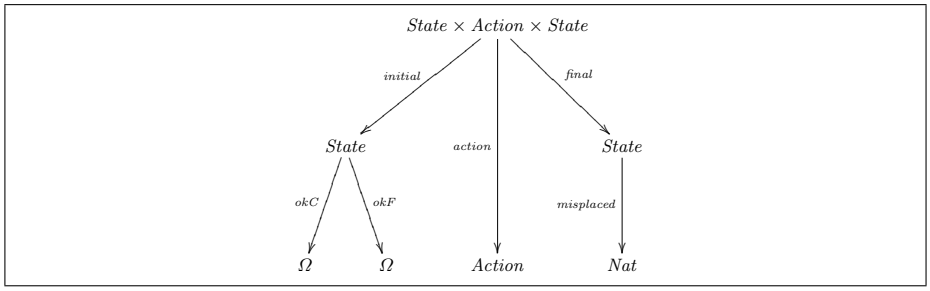

Fig. 8. Evidence and result variables for the blocks world agent

Figure 9 gives the influence diagram for the blocks world agent. Note that, in this illustration, the utility of a state is determined by a single result variable.

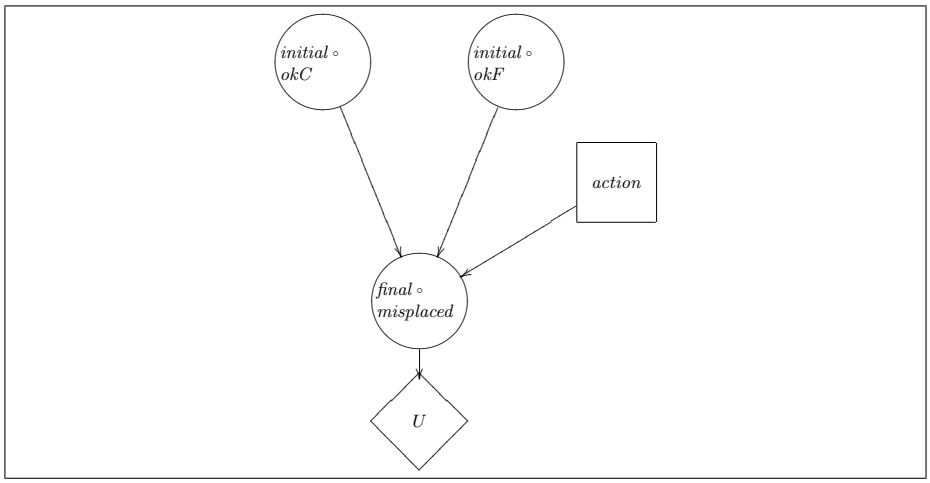

Fig. 9. Influence diagram for the blocks world agent

Figure 10 shows the conditioned influence diagram (where actual counts are shown instead of probabilities; these counts need to be normalised). The observations are from 100 episodes in a world of 10 blocks for which each episode involved achieving an intention with 10 on-states (that is, the each goal state was fully specified). There are only four rows in this table instead of eight because in four cases the precondition of the corresponding action is not satisfied; in each of the four rows that remain, the precondition of the corresponding action is satisfied. The counts in this table were obtained by deploying the agent and observing the effects of actions on various states. The utilities are based on a value of 1 if there are no misplaced blocks (that is, a goal state has been reached) with a discount factor of 0.9 for every further misplaced block.

In this very simple case, the only choice about actions comes in the case that the preconditions of both actions are satisfied. This is shown in the first two rows of Figure 10. Applying the principle of maximum expected utility to these two rows leads to the action *CMove* being selected. The policy is given in Figure 11 and its definition in the logic is as follows.

$$
\begin{array}{l}
policy : State \rightarrow Action \\
\Box_t \, \forall_{State} s. \\
\quad ((policy \; s) =_{Action} \\
\quad\quad if \; ((okC \; s) =_{\Omega} \top) \; then \; CMove \\
\quad\quad else \; FMove).
\end{array}
$$

		final ∘ *misplaced*											
initial ∘ (okC, okF)	*action*	0	1	2	3	4	5	6	7	8	9	10	
$(\top, \top)$	*CMove*	0	1	1	4	14	31	42	53	63	38	0	0.49
$(\top, \top)$	*FMove*	0	0	0	1	4	18	23	69	58	58	11	0.46
$(\top, \bot)$	*CMove*	100	99	99	96	85	66	52	24	8	0	0	0.76
$(\bot, \top)$	*FMove*	0	0	0	2	3	4	9	35	104	115	117	0.40
		1.0	0.9	0.81	0.73	0.66	0.59	0.53	0.48	0.43	0.39	0.35	

Fig. 10. Influence diagram conditioned on the evidence and action variables for the blocks world agent

initial ∘ (okC, okF)	*action*
$(\top, \top)$	*CMove*
$(\top, \bot)$	*CMove*
$(\bot, \top)$	*FMove*

Fig. 11. Policy for the blocks world agent

4.3 Comparision of the Illustrations

Between them, the two illustrations highlight most of the important features of the rational agent architecture.

First, note that in each illustration we have brought expert knowledge to bear to make good choices of the state, actions, evidence and result variables, and utilities. For example, for blocks world, we used knowledge of the domain to define the action *CMove*. While it would be possible to pretend not to have expert knowledge about the domain and use more primitive actions instead, our view is that, faced with a real-world application, it is natural to leverage every piece of expert knowledge about the domain that is available. We have followed this approach in both illustrations.

The TV recommender provides a good illustration of evidence variables that are acquired, in this case, from the user. They were chosen because their values provide an evidence tuple that is a good guide to the recommendation that the agent should make. Their definitions are quite complicated and user-specific. (See [2] for details about *ultp*.) On the other hand, in blocks world, the evidence variables are as simple as they can possibly be – just the preconditions for the actions. This situation would be untypical for more complex applications, but seems reasonable for an agent whose main task is to move in a search space.

For the TV recommender we have the advantage of an oracle – the user – who can directly criticise its recommendations. Thus the form of the result variables in that illustration seems natural. For blocks world, we exploit the fact that it is possible to make a fairly accurate estimate of the length of the path from the current state to the goal. This naturally suggests that *misplaced* should be the result variable.

In blocks world, the agent may have to make many moves to reach the goal. In other words, a plan to reach the goal emerges from the policy that is just concerned with 'locally' choosing the best action. For real-world applications, it is typical for several actions to be needed to accomplish a task and it is also typical for actions to have non-deterministic outcomes, as in blocks world.

Notwithstanding the positive aspects of the illustrations, neither provides fully convincing evidence of the usefulness of the rational agent architecture. Studying more complex applications is a perhaps the most important next step.

5 Discussion

We conclude with some remarks about the wider context of this research.

The scientific question of interest here is that of designing effective architectures for agent systems that perform complex tasks. We take the view that a satisfactory solution to this problem will involve (at least) a successful integration of logic, probability and machine learning: logic is need for knowledge representation and reasoning, probability is needed for handling uncertainty, and machine learning is needed for adaptivity. The integration of logic and probability is an old scientific problem going back to the 19th Century that computer scientists and others are still struggling with today. Recently, the goal of integrating logic, probability, and learning was explicitly identified as a major research problem. (An excellent survey of this problem is given in [7]. See also the Dagstuhl Seminar on this topic at `http://www.dagstuhl.de/05051/`.) Of course, there are many possible approaches to solving this problem, quite a few of which were presented at the Dagstuhl Seminar and most of which are primarily set in a machine learning context. Here we take the view that the best general setting for this problem is that of agents.

The main elements of our approach to the architecture of agents are as follows. First, there are the agent beliefs. For these, we employ a modal higher-order logic. While neither of the illustrations in this paper make truly crucial use of the modalities, we have applications in mind where these would be essential. In general, we expect to make use of both epistemic modalities, as in this paper, and temporal modalities, as these are the two kinds of modalities that seem most useful for agents. The beliefs are maintained as function definitions of evidence and result variables. There is a general method for acquiring beliefs that includes conventional database update and machine learning as special cases.

The other main element of the architecture is a Bayesian network on the values of evidence and result variables. The parameters for this Bayesian network are learned through training. These parameters together with the utilities on result tuples determine the action with the highest expected utility for each evidence tuple and, therefore, the policy for the agent. Thus our approach has integrated logic and probability in the following way: logic is used to provide the definitions of the evidence and result variables, while probability is used to provide distributions on the values of these variables. Machine learning is needed in two places in the architecture: the definitions of some variables are

learned by belief acquisition and the Bayesian network parameters are learned by maximum-likelihood estimation.

Our approach to agent construction assumes that expert knowledge is available for each application. Thus, given an application, it must be possible to know what should be the state, actions, definitions (or, at least, hypothesis languages) for the evidence and result variables, and utilities of the agent. With these in place, the agent policy can be learned through training. Our experience with a small number of application domains suggests a methodology for constructing agents by our approach can be successfully developed, although much work on this remains to be done.

Currently, work is proceeding on the theoretical foundations, architectural issues, and applications. Particularly important are the applications as we need to deploy agents in application areas rather more complex than those studied in this paper to make the whole approach fully convincing. One particularly promising application area that we are investigating is a poker-playing program. This application is a challenging one and provides a good testbed for agent and AI technologies. In particular, it exercises every aspect of the rational agent architecture.

Acknowledgements

The authors are grateful to Joshua Cole and Kee Siong Ng for many helpful discussions that have greatly influenced the ideas in this paper and for their implementation of the agent applications reported here.

References

1. C. Boutilier, T. Dean, and S. Hanks. Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.
2. J.J. Cole, M.J. Gray, J.W. Lloyd, and K.S. Ng. Personalisation for user agents. In *Fourth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 05)*, 2005.
3. R.G. Cowell, A.P. Dawid, S.L. Lauritzen, and D.J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Statistics for Engineering and Information Sciences. Springer, 1999.
4. J.W. Lloyd. *Logic for Learning*. Cognitive Technologies. Springer, 2003.
5. J.W. Lloyd. Modal higher-order logic for agents. `http://users.rsise.anu.edu.au/~jwl/beliefs.pdf`.
6. J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.
7. Luc De Raedt and Kristian Kersting. Probabilistic logic learning. *SIGKDD Explorations*, 5(1):31–48, 2003.
8. R. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.
9. S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, second edition, 2002.
10. M. Wooldridge. *Reasoning about Rational Agents*. MIT Press, 2000.