Probabilistic and Logical Beliefs

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

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

Abstract. This paper proposes a method of integrating two different concepts of belief in artificial intelligence: belief as a probability distribution and belief as a logical formula. The setting for the integration is a highly expressive logic. The integration is explained in detail, as its comparison to other approaches to integrating logic and probability. An illustrative example is given to motivate the usefulness of the ideas in agent applications.

1 Introduction

The term 'belief' has two meanings in artificial intelligence: in robotics and vision [1], a 'belief' is generally a probability distribution; in logical artificial intelligence, a 'belief' is a logical formula. In this paper, we give a definition of belief that encompasses both meanings and investigate the use of this concept for agent applications.

This work is set in the context of the more general problem of integrating logic and probability, a problem of major importance in artificial intelligence that is currently attracting substantial interest [2–6, **?**]. Consequently, to set the scene and also to provide a contrast with the approach to integration adopted in this paper, we now briefly discuss the most common approach in literature.

Unfortunately, there does not seem to be any widely agreed statement of exactly what the problem of integrating logic and probability actually is, much less a widely agreed solution to the problem [7–10]. However, the following quote from [9], which contains an excellent overview of the problem especially from the philosophical point of view, captures the generally agreed essence of the problem: "Classical logic has no explicit mechanism for representing the degree of certainty of premises in an argument, nor the degree of certainty in a conclusion, given those premises". Thus, intuitively, the problem is to find some way of effectively doing probabilistic reasoning in a logical formalism that may involve the invention of 'probabilistic logics'. The discussion below is restricted to recent approaches that have come from the artificial intelligence community; these approaches usually also include a significant component of learning [10].

The standard logical setting for these approaches is first-order logic. Imagine that an agent is operating in some environment for which there is some uncertainty (for example, the environment might be partially observable). The environment is modelled as a probability distribution over the collection of first-order interpretations (over some suitable alphabet for the application at hand). The intuition is that any of these interpretations could be the actual environment but that some interpretations are more likely than others to correctly model the actual world and this information is given by the distribution on the interpretations. If the agent actually knew this distribution, then it could answer probabilistic questions of the form: if (closed) formula ψ holds, what is the probability that the (closed) formula φ holds? In symbols, the question is: what is $Pr(\varphi \,|\, \psi)$?

We formalise this situation. Let $\mathfrak{I}$ be the set of interpretations and p a probability measure on the σ-algebra of all subsets of this set. Define the random variable $X_\varphi : \mathfrak{I} \to \mathbb{R}$ by

$$X_\varphi(I) = \begin{cases} 1 & \text{if } \varphi \text{ is true in } I \\ 0 & \text{otherwise,} \end{cases}$$

with a similar definition for X_ψ. Then $Pr(\varphi \,|\, \psi)$ can be written in the form

$$p(X_\varphi = 1 \,|\, X_\psi = 1)$$

which is equal to

$$\frac{p(X_\varphi = 1 \wedge X_\psi = 1)}{p(X_\psi = 1)}$$

and, knowing p, can be evaluated.

Of course, the real problem is to know the distribution on the interpretations. To make some progress on this, most systems intending to integrate logical and probabilistic reasoning in artificial intelligence make simplifying assumptions. For a start, most are based on Prolog. Thus theories are first-order Horn clause theories, maybe with negation as failure. Interpretations are limited to Herbrand interpretations and often function symbols are excluded so the Herbrand base (and therefore the number of Herbrand interpretations) is finite. Let $\mathfrak{I}$ denote the (finite) set of Herbrand interpretations and $\mathcal{B}$ the Herbrand base. We can identify $\mathfrak{I}$ with the product space $\{0,1\}^{\mathcal{B}}$ in the natural way. Thus the problem amounts to knowing the distribution on this product space. At this point, there is a wide divergence in the approaches. For example, either Bayesian networks or Markov random fields can be used to represent the product distribution. In [4], the occurrences of atoms in the same clause are used to give the arcs and the weights attached to clauses are used to give the potential functions in a Markov random field. In [6], conditional probability distributions are attached to clauses to give a Bayesian network. In [3], a program is written that specifies a generative distribution for a Bayesian network. In all cases, the logic is exploited to give some kind of compact representation of what is usually a very large graphical

model. Generally, the theory is only used to construct the graphical model and reasoning proceeds probabilistically, as described above.

Here we follow a different approach. To begin with, we use a much more expressive logic, modal higher-order logic. The higher-orderness will be essential to achieve the desired integration of logic and probability. Also, the modalities will be important for agent applications. Furthermore, in our approach, the theory plays a central role and probabilistic reasoning all takes place in the context of the theory.

A full account of the logic we employ is given in [11]; much briefer accounts can be found in [12, 13]. For lack of space, the reader is referred to these for the details. We assume there are necessity modality operators $\Box_i$, for $i = 1, \ldots, m$. As is well known, modalities can have a variety of meanings, depending on the application. 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. The symbol $\Box$ denotes a sequence of modalities. The type of the booleans is denoted by Ω, the type of the integers by *Int*, and the type of the reals by *Real*. Also $(List\ \sigma)$ is the type of lists whose items have type σ and $\{\sigma\}$ is the type of sets whose elements have type σ. Composition is handled by the (reverse) composition function $\circ$ defined by $((f \circ g)\ x) = (g\ (f\ x))$.

The next section defines the central concept of a density and gives some of its properties. Section 3 presents our approach to integrating logic and probability. Section 4 considers the idea that beliefs should be function definitions. Section 5 gives an extended example to illustrate the ideas. Section 6 gives some conclusions and future research directions.

2 Densities

This section presents some standard notions of measure theory, particularly that of a density, which will be needed later.

Definition 1. *Let $(X, \mathcal{A}, \mu)$ be a measure space and $f : X \to \mathbb{R}$ a measurable function. Then f is a* density (wrt the measure μ) *if (i) $f(x) \geq 0$, for all $x \in X$, and (ii) $\int_X f\, d\mu = 1$.*

There are two main cases of interest. The first is when μ is the counting measure on X, in which case $\int_X f\, d\mu = \sum_{x \in X} f(x)$; this is the discrete case. The second case is when X is $\mathbb{R}^n$, for some $n \geq 1$, and μ is Lebesgue measure; this is the continuous case.

A density f gives a probability ν on $\mathcal{A}$ by the definition

$$\nu(A) = \int_A f \, d\mu,$$

for $A \in \mathcal{A}$. In the common discrete case, this definition specialises to

$$\nu(A) = \sum_{x \in A} f(x).$$

We let *Density* X be a synonym for $X \to \mathbb{R}$, but with the understanding that functions in *Density* X are intended to be densities, not just arbitrary real-valued functions. Some (higher-order) functions that operate on densities will be needed. The following gives the natural 'composition' $\natural$ of two functions whose codomains are densities.

Definition 2. *Let* $(X, \mathcal{A}, \mu)$, $(Y, \mathcal{B}, \nu)$, *and* $(Z, \mathcal{C}, \xi)$ *be measure spaces. The function* $\natural : (X \to \textit{Density } Y) \to (Y \to \textit{Density } Z) \to (X \to \textit{Density } Z)$ *is defined by*

$$(f \natural g)(x)(z) = \int_Y f(x)(y) \times g(y)(z) \, d\nu(y),$$

for $f : X \to \textit{Density } Y$, $g : Y \to \textit{Density } Z$, $x \in X$, *and* $z \in Z$.

Specialised to the discrete case, the definition is

$$(f \natural g)(x)(z) = \sum_{y \in Y} f(x)(y) \times g(y)(z).$$

The composition of two functions, the first of which has a codomain of densities can be similarly defined. Also, the case when the codomain of the first function is the same as the domain of the second function can be handled by the usual composition of functions.

We can define conditional densities. Consider a function $f : \textit{Density } X \times Y$ that defines a product density. Then we can express the conditional density obtained by conditioning on values in X by the function $f_1 : X \to \textit{Density } Y$ defined by

$$f_1(x)(y) = \frac{f(x,y)}{\int_Y f(x,y) \, d\nu(y)},$$

for $x \in X$ and $y \in Y$. Clearly, $f_1(x)$ *is* a density. Conditioning on the other argument is analogous to this.

Marginal densities can also be defined. Consider a function

$$f : \textit{Density } X \times Y \times Z$$

that defines a product density. Then we can form the marginal density over the first argument by the function $f_1 : \textit{Density } X$ defined by

$$f_1(x) = \int_Z \int_Y f(x,y,z) \, d\nu(y) \, d\xi(z),$$

for $x \in X$. By Fubini's theorem, f_1 *is* a density. This is easily extended to marginalising in arbitrary products.

3 Integrating Logic and Probability

This section provides an overview of our approach to integrating logic and probability. The key idea is to allow densities to appear in theories. For this reason, we first set up some logical machinery for this. In the logic, we introduce the type synonym $\mathit{Density}\ a \equiv a \to \mathit{Real}$ with the same motivation as in the previous section. (Here a is a type variable; so $\mathit{Density}\ a$ is a polymorphic type.) Any term of type $\mathit{Density}\ \tau$, for some τ, is called a *density*. We also make available the functions from Section 2 that compose functions whose codomains are densities. Conditionalisation and marginalisation are also easily expressed.

The idea is to model uncertainty by using densities in the definitions of (some) functions in theories. Consider a function $f : \sigma \to \tau$ for which there is some uncertainty about its values that we want to model. We do this with a function

$$f' : \sigma \to \mathit{Density}\ \tau,$$

where, for each argument t, $(f'\ t)$ is a suitable density for modelling the uncertainty in the value of the function $(f\ t)$. The intuition is that the actual value of $(f\ t)$ is likely to be where the 'mass' of the density $(f'\ t)$ is most concentrated. Of course, (unconditional) densities can also be expressed by functions having a signature of the form $\mathit{Density}\ \tau$.

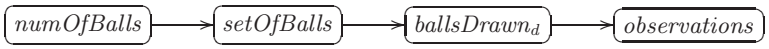
This simple idea turns out to be a powerful and convenient way of modelling uncertainty with logical theories in diverse applications, especially agent applications. Note carefully the use that has been made of the expressive logic here. Functions whose values are densities are higher-order functions that cannot be modelled directly in first-order logic.

As well as representing knowledge, it is necessary to reason with it. We employ a declarative programming language called Bach for this purpose. Bach is a probabilistic modal functional logic programming language. Programs in the language are equational theories in modal higher-order logic. The reasoning system for the logic underlying Bach combines a theorem prover and an equational reasoning system [11, 13, 14]. The theorem prover is a fairly conventional tableau theorem prover for modal higher-order logic. The equational reasoning system is, in effect, a computational system that significantly extends existing declarative programming languages by adding facilities for computing with modalities and densities. 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 are two examples to illustrate the ideas introduced so far.

Example 1. We model the following scenario, which is one of the main examples used in [3]. An urn contains an unknown number of balls that have the colour blue or green with equal probability. Identically coloured balls are indistinguishable. An agent has the prior belief that the distribution of the number of balls is a Poisson distribution with mean 6. The agent now draws some balls from the urn, observes their colour, and then replaces them. The observed colour is different from the actual colour of the ball drawn with probability 0.2. On the basis of these observations, the agent should infer certain properties about the urn, such as the number of balls it contains.

It was claimed in [3] that this problem cannot be modelled in most existing first-order probabilistic languages because the number of balls in the urn is unknown. Interestingly, the problem can be modelled rather straightforwardly if we define densities over structured objects like sets and lists. The following is a suitable graphical model.



In the simulation given by the following Bach program, a number n is selected from the Poisson distribution and a set s of balls of size n is constructed. A ball is represented by an integer identifier and its colour: $Ball = Int \times Colour$. The balls in s are labelled 1 to n, and the colours are chosen randomly. Given s, a list is constructed consisting of d balls by drawing successively at random with replacement from s. The observed colours of the drawn balls are then recorded.

$$
\begin{aligned}
&\mathit{colour} : \mathit{Colour} \to \Omega \\
&(\mathit{colour}\ x) = (x = \mathit{Blue}) \vee (x = \mathit{Green}) \\
&\mathit{numOfBalls} : \mathit{Density}\ \mathit{Int} \\
&(\mathit{numOfBalls}\ x) = (\mathit{poisson}\ 6\ x) \\
&\mathit{poisson} : \mathit{Int} \to \mathit{Density}\ \mathit{Int} \\
&(\mathit{poisson}\ x\ y) = e^{-x}x^y/y! \\
&\mathit{setOfBalls} : \mathit{Int} \to \mathit{Density}\ \{\mathit{Ball}\} \\
&(\mathit{setOfBalls}\ n\ s) = \mathit{if}\ \exists x_1 \cdots \exists x_n.((\mathit{colour}\ x_1) \wedge \cdots \wedge (\mathit{colour}\ x_n) \wedge \\
&\qquad\qquad (s = \{(1, x_1), \ldots, (n, x_n)\})\ \mathit{then}\ 0.5^n\ \mathit{else}\ 0 \\
&\mathit{ballsDrawn} : \mathit{Int} \to \{\mathit{Ball}\} \to \mathit{Density}\ (\mathit{List}\ \mathit{Ball}) \\
&(\mathit{ballsDrawn}\ d\ s\ x) = \\
&\quad \mathit{if}\ \exists x_1 \cdots \exists x_d.((s\ x_1) \wedge \cdots \wedge (s\ x_d) \wedge (x = [x_1, \ldots, x_d]))\ \mathit{then}\ (\mathit{card}\ s)^{-d}\ \mathit{else}\ 0 \\
&\mathit{observations} : (\mathit{List}\ \mathit{Ball}) \to \mathit{Density}\ (\mathit{List}\ \mathit{Colour}) \\
&(\mathit{observations}\ x\ y) = \mathit{if}\ (\mathit{length}\ x) = (\mathit{length}\ y)\ \mathit{then}\ (\mathit{obsProb}\ x\ y)\ \mathit{else}\ 0 \\
&\mathit{obsProb} : (\mathit{List}\ \mathit{Ball}) \to (\mathit{List}\ \mathit{Colour}) \to \mathit{Real} \\
&(\mathit{obsProb}\ []\ []) = 1
\end{aligned}
$$

$$(obsProb\ (\#\ (x_1, y_1)\ z_1)\ (\#\ y_2\ z_2)) =$$
$$(if\ (y_1 = y_2)\ then\ 0.8\ else\ 0.2) \cdot (obsProb\ z_1\ z_2)$$

$$joint : Int \to Density\ (Int \times \{Ball\} \times (List\ Ball) \times (List\ Colour))$$
$$(joint\ d\ (n, s, x, y)) =$$
$$(numOfBalls\ n) \cdot (setOfBalls\ n\ s) \cdot (ballsDrawn\ d\ s\ x) \cdot (observations\ x\ y)$$

The function *card* returns the cardinality of a set and *length* returns the size of a list. The functions *setOfBalls* and *ballsDrawn* are defined informally above; formal recursive definitions can be given.

Marginalisations and conditionalisations of the given density can be computed to obtain answers to different questions. For example, the following gives the probability that the number of balls in the urn is m after the colours $[o_1, o_2, \ldots, o_d]$ from d draws have been observed:

$$\frac{1}{K} \sum_s \sum_l (joint\ d\ (m, s, l, [o_1, o_2, \ldots, o_d])),$$

where K is a normalisation constant, s ranges over $\{\, s \mid (setOfBalls\ m\ s) > 0 \,\}$, and l ranges over $\{\, l \mid (ballsDrawn\ d\ s\ l) > 0 \,\}$. The elements of these two sets are automatically enumerated by Bach during execution of the query.

Figure 1 shows the posterior distribution of m, the actual number of balls in the urn, after drawing ten and fifteen balls, and observing that they are all blue. The Poisson curve is the prior distribution of m. Consistent with intuition, the lower numbers of m become increasingly probable (compared to the prior distribution) as more and more blue balls are observed.

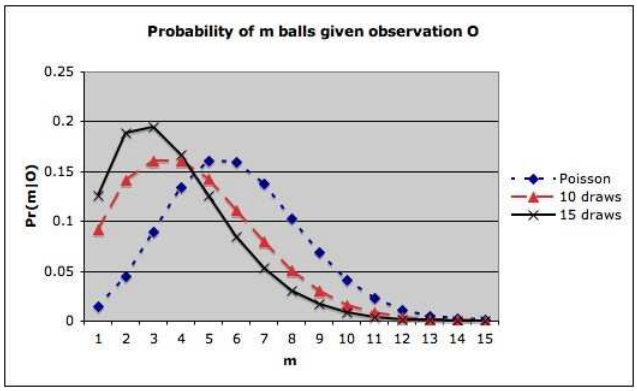
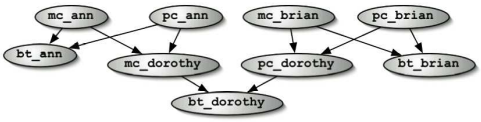


Fig. 1. Posterior distribution of the number of balls

Example 2. We next look at an application taken from [6] (the problem was introduced earlier in [15]). It is a genetic model of the inheritance of a single

gene that determines a person's blood type. Each person has two copies of the chromosome containing this gene, one, the m-chromosome, inherited from her mother, and one, the p-chromosome, inherited from her father. The following figure from [6] shows a Bayesian network modelling the inheritance of blood types within a particular family.



Obviously, such a network can be written down and reasoned with. But note that although different families are associated with different Bayesian networks, all these networks share essentially the same basic structure. By separating the descriptions of family-dependent and -independent parts, logic can be used to compactly represent and reason with a large class of similar Bayesian networks. This is the strategy employed by systems like BLP [6].

We now show one way the Bayesian logic program given in [6, Fig. 1.4] can be coded in Bach. We define a probability distribution over triples (l_1, l_2, l_3), where l_1 and l_2 are lists of chromosomes representing the m- and p-chromosomes of each person in the family, and l_3 is a list representing each person's blood type. The ith entry in each list refers to the ith person in the list $family : List\ Person$. By changing the definitions of

$$family,\ \ mother : Person \to Person \ \text{ and } \ father : Person \to Person,$$

we get different Bayesian networks for different families.

$$family = [Ann, Brian, Dorothy]$$
$$(mother\ x) = if\ (x = Dorothy)\ then\ Ann\ else\ Unknown$$
$$(father\ x) = if\ (x = Dorothy)\ then\ Brian\ else\ Unknown$$

$$joint : Density\ (List\ Chromosome) \times (List\ Chromosome) \times (List\ BloodType)$$
$$(joint\ (x_1, x_2, x_3)) = (joint_2\ family\ (x_1, x_2, x_3)\ (x_1, x_2))$$

$$joint_2 : (List\ Person) \to (List\ Chromosome) \times (List\ Chromosome) \times$$
$$(List\ BloodType) \to (List\ Chromosome) \times (List\ Chromosome) \to Real$$
$$(joint_2\ []\ ([], [], [])\ l) = 1$$
$$(joint_2\ (\#\ p\ t_0)\ ((\#\ x_1\ t_1), (\#\ x_2\ t_2), (\#\ x_3\ t_3))\ l) =$$
$$(mc\ p\ l\ x_1) \cdot (pc\ p\ l\ x_2) \cdot (bt\ x_1\ x_2\ x_3) \cdot (joint_2\ t_0\ (t_1, t_2, t_3)\ l)$$

$$mc : Person \to (List\ Chromosome) \times (List\ Chromosome) \to Chromosome \to Real$$
$$(mc\ p\ (l_1, l_2)\ x) = if\ (mother\ p) = Unknown\ then\ 1/3$$
$$else\ (mc_2\ (getVal\ (mother\ p)\ l_1)\ (getVal\ (mother\ p)\ l_2)\ x)$$

$$mc_2 : Chromosome \to Chromosome \to (Density\ Chromosome)$$

$(mc_2\ A\ A\ z) = \mathit{if}\ (z = A)\ \mathit{then}\ 0.98\ \mathit{else}\ 0.01 \quad \ldots$ etc

$bt : \mathit{Chromosome} \to \mathit{Chromosome} \to (\mathit{Density}\ \mathit{BloodType})$
$(bt\ A\ A\ z) = \mathit{if}\ (z = A)\ \mathit{then}\ 0.97\ \mathit{else}\ 0.01 \quad \ldots$ etc

The function *getVal* returns the nth element in a list indexed by a person. The function *pc* is defined in a similar way to *mc*. The functions *bt* and mc_2 are reproductions of the two conditional probability tables given in [6, Fig. 1.4]. It is assumed that all the arguments to the function *joint* have the same length.

Just as in [6] and [15], the m-chromosome, p-chromosome, and blood type of each person is a random variable in the given joint distribution.

Given the above, one can answer questions like "What is the probability that Dorothy has blood type A given that the m-chromosome of *Brian* is A and the p-chromosome of *Ann* is O?" by computing

$$\frac{\sum_{x_i,y_i,z_i}(\mathit{joint}\ ([x_1, A, x_2], [O, y_1, y_2], [z_1, z_2, A]))}{\sum_{x_i,y_i,z_i}(\mathit{joint}\ ([x_1, A, x_2], [O, y_1, y_2], [z_1, z_2, z_3]))},$$

where $x_i, y_i : \mathit{Chromosome}$ and $z_i : \mathit{BloodType}$.

At this stage, it is interesting to make a comparison with other approaches to integrating logic and probability. Perhaps the main point is the value of working in a higher-order logic. All other logical approaches to this integration that we know of use first-order logic and thereby miss the opportunity of being able to reason about densities in theories. This is an important point. (Classical) logic is often criticised for its inability to cope with uncertainty: witness the quote in the introduction. We believe this view is simply wrong! – higher-order logic is quite capable of modelling probabilistic statements about knowledge directly in theories themselves, thus providing a powerful method of capturing uncertainty. In first-order logic, there is a preoccupation with the truth or falsity of formulas, which does seem to preclude the possibility of capturing uncertainty. However, looked at from a more general perspective, first-order logic is impoverished. It is not natural to exclude higher-order functions – these are used constantly in everyday (informal) mathematics. Also the rigid dichotomy between terms and formulas in first-order logic gets in the way. In higher-order logic, a formula is a term whose type just happens to be boolean; also it is just as important to compute the value of arbitrary terms, not only formulas. Higher-order logic is essentially the language of everyday mathematics and no-one would ever claim situations involving uncertainty and structural relationships between entities cannot be modelled directly and in an integrated way using mathematics – therefore they can also be so modelled using higher-order logic.

Another significant difference concerns the semantic view that is adopted. In the most common approach to integration explained above there is assumed to be a distribution on interpretations and answering queries involves performing computations over this distribution. In principle, this is fine; given the distribution, one can answer queries by computing with this distribution. But this approach is intrinsically more difficult than computing the value of terms in the

traditional case of having *one* intended interpretation, the difficulty of which has already led to nearly all artificial intelligence systems using the axiomatic approach of building a theory (that has the intended interpretation as a model) and proving theorems with this theory instead. Here we adopt the axiomatic method of using a theory to model a situation and relying on the soundness of theorem proving to produce results that are correct in the intended interpretation [11]. We simply have to note that this theory, if it is higher-order, can include densities that can be reasoned with. *Thus no new conceptual machinery at all needs to be invented.* In our approach, whatever the situation, there is a single intended interpretation, which would include densities in the case where uncertainty is being modelled, that is a model of the theory. Our approach also gives fine control over exactly what uncertainty is modelled – we only introduce densities in those parts of the theory that really need them. Furthermore, the probabilistic and non-probabilistic parts of a theory work harmoniously together.

4 Beliefs

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 3. *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*.*

The function f *thus defined is called a* belief function*. In case* τ *has the form Density* ν*, for some* ν*, we say the belief is* probabilistic*.*

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 [16] to motivate the introduction of Definition 3 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 2, 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.

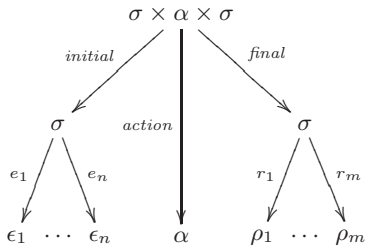


Fig. 2. Evidence and result features

Now consider the function

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

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 $\mathit{policy} : \sigma \rightarrow \alpha$ is then defined by

$$(\mathit{policy}\ s) = \arg\max_a \mathbb{E}_{(\mathit{transition}\ ((e_1\ s),\ldots,(e_n\ s),a))}(\mathit{utility}),$$

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 $\mathbb{E}_{(\mathit{transition}\ ((e_1\ s),\ldots,(e_n\ s),a))}$ denotes the expectation with respect to the density $(\mathit{transition}\ ((e_1\ s),\ldots,(e_n\ s),a))$.

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 3.

The above description generalises the agent architecture presented in [16] by admitting probabilistic beliefs in addition to non-probabilistic ones. Using a density to model the uncertain value of a function on some argument is better than using a single value (such as the mean of the density). For example, if the density is a normal distribution, then it may be important that the variance is large or small: if it is large, intuitively, there is less confidence about its actual value; if it is small, then there could be confidence that the actual value is the mean. Such subtleties can assist in the selection of one action over another. Similarly, learning tasks can exploit the existence of the density by including features based on the mean, variance, higher moments, or other parameters of the density in hypothesis languages.

We now examine the form 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. In particular, some of these beliefs may be compositions of other probabilistic beliefs in which case the composition operators introduced in Section 2 will be useful. 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.

$$
\begin{aligned}
&\Box\forall x.((f\ x) = \\
&\quad \mathit{if}\ (p_1\ x)\ \mathit{then}\ v_1 \\
&\quad \mathit{else\ if}\ (p_2\ x)\ \mathit{then}\ v_2 \\
&\qquad \vdots \\
&\quad \mathit{else\ if}\ (p_n\ x)\ \mathit{then}\ v_n \\
&\quad \mathit{else}\ v_0),
\end{aligned}
\tag{1}
$$

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 $\Box$. Note that in the case when τ has the form *Density* ν, for some ν, the values $v_0, v_1, \ldots, v_n$ are densities.

While the above form for acquired beliefs may appear to be rather specialised, it turns out to be convenient and general, and easily encompasses beliefs in many other forms [17, 18]. Also the decision-list form of the definitions is highly convenient for acquisition using some kind of learning algorithm [12, 19–21]. Towards that end, the Alkemy machine learning system [19, 20] is being extended with capability to acquire modal and probabilistic beliefs.

5 Illustration

Here is an extended example to illustrate the ideas that have been introduced.

Example 3. 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 in the form of beliefs about the function *likes* : *Program* $\rightarrow$ *Density* Ω. 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:

$$\boldsymbol{B}_m\boldsymbol{B}_a\,\forall x.((\mathit{likes}\ x) = \varphi) \tag{2}$$

$$\boldsymbol{B}_m\boldsymbol{B}_b\,\forall x.((\mathit{likes}\ x) = \psi) \tag{3}$$

$$\boldsymbol{B}_m\boldsymbol{B}_c\,\forall x.((\mathit{likes}\ x) = \xi) \tag{4}$$

for suitable φ, ψ, and ξ. We will now look at the form of a belief about *likes*. Methods for acquiring these beliefs were studied in [22].

Figure 3 is a typical definition acquired incrementally using real data. In the beginning, the belief base contains the formula

$$\boldsymbol{B}_m\blacksquare\boldsymbol{B}_a\forall x.((\mathit{likes}\ x) = \lambda y.\mathit{if}\ (y = \top)\ \mathit{then}\ 0.5\ \mathit{else\ if}\ (y = \bot)\ \mathit{then}\ 0.5\ \mathit{else}\ 0).$$

The meaning of this formula is "the agent believes that, at all times in the past, Alice has no preference one way or another over any program". After 3 time steps, this formula has been transformed into the last formula in Figure 3. 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. Note how useful parts of previously acquired beliefs are recycled in forming new beliefs.

We have seen the general form of beliefs (2)-(4). Given these beliefs about the occupant preferences for TV programs, the task for the majordomo agent is to recommend programs that all three occupants would be interested in watching together. To estimate how likely the group as a whole likes a program, the agent simply counts the number of positive preferences, leading to the definition of *aggregate* below. To deal with the fact that user preferences are not definite but can only be estimated, we compose *aggregate* with the function *combinePrefs* to form *groupLikes*, where *combinePrefs* brings the individual preferences together.

$$\begin{aligned}&\mathit{combinePrefs} : \mathit{Program} \rightarrow \mathit{Density}\ \Omega \times \Omega \times \Omega\\ &\boldsymbol{B}_m\,\forall p.\forall x.\forall y.\forall z.((\mathit{combinePrefs}\ p\ (x,y,z)) =\\ &\quad (\boldsymbol{B}_a\mathit{likes}\ p\ x)\cdot(\boldsymbol{B}_b\mathit{likes}\ p\ y)\cdot(\boldsymbol{B}_c\mathit{likes}\ p\ z))\end{aligned}$$

$\boldsymbol{B}_m\, \boldsymbol{B}_a\, \forall x.((likes\ x) =$
$\quad if\ (projTitle \circ (=\ \text{“}NFL\ Football\text{”})\ x)\ then\ \lambda y.if\ (y = \top)\ then\ 1\ else\ if\ (y = \bot)\ then\ 0\ else\ 0$
$\quad else\ if\ (projTitle \circ (existsWord\ (=\ \text{“}sport\text{”}))\ x)\ then\ \lambda y.if\ (y = \top)\ then\ 0.7$
$\qquad\qquad else\ if\ (y = \bot)\ then\ 0.3\ else\ 0$
$\quad else\ (\bullet likes\ x))$
$\bullet \boldsymbol{B}_m\, \boldsymbol{B}_a\, \forall x.((likes\ x) =$
$\quad if\ (projGenre \circ (=\ Documentary)\ x)\ then\ \lambda y.if\ (y = \top)\ then\ 0.9\ else\ if\ (y = \bot)\ then\ 0.1\ else\ 0$
$\quad else\ if\ (projGenre \circ (=\ Movie)\ x)\ then\ \lambda y.if\ (y = \top)\ then\ 0.75\ else\ if\ (y = \bot)\ then\ 0.25\ else\ 0$
$\quad else\ (\bullet likes\ x))$
$\bullet^2 \boldsymbol{B}_m\, \boldsymbol{B}_a\, \forall x.((likes\ x) =$
$\quad if\ (projGenre \circ (=\ Documentary)\ x)\ then\ \lambda y.if\ (y = \top)\ then\ 1\ else\ if\ (y = \bot)\ then\ 0\ else\ 0$
$\quad else\ if\ (projGenre \circ (=\ Drama)\ x)\ then\ \lambda y.if\ (y = \bot)\ then\ 0.8\ else\ if\ (y = \top)\ then\ 0.2\ else\ 0$
$\quad else\ (\bullet likes\ x))$
$\bullet^3 \boldsymbol{B}_m \blacksquare \boldsymbol{B}_a \forall x.((likes\ x) = \lambda y.if\ (y = \top)\ then\ 0.5\ else\ if\ (y = \bot)\ then\ 0.5\ else\ 0).$

Fig. 3. Part of the belief base of the agent

$$aggregate : \Omega \times \Omega \times \Omega \to Density\ \Omega$$
$$\boldsymbol{B}_m\, \forall x.\forall y.\forall z.((aggregate\ (x, y, z)) =$$
$$\lambda v.\frac{1}{3}((\mathbb{I}\ (x = v)) + (\mathbb{I}\ (y = v)) + (\mathbb{I}\ (z = v))))$$
$$groupLikes : Program \to Density\ \Omega$$
$$\boldsymbol{B}_m\, \forall x.((groupLikes\ x) =$$
$$((\natural\ combinePrefs\ aggregate)\ x)).$$

Here, $\mathbb{I} : \Omega \to Int$ is the indicator function defined by $(\mathbb{I}\ \top) = 1$ and $(\mathbb{I}\ \bot) = 0$.

The version of *combinePrefs* given above makes an independence assumption amongst the densities $\boldsymbol{B}_a likes\ p$, and so on. It may be that there are some dependencies between these densities that could be learned. In this case, a more complicated definition of *combinePrefs* would be substituted for the one above. Analogous comments apply to *aggregate*.

Now let us look more closely at what the architecture of the agent would look like if the rational agent architecture were used for this task. The function *groupLikes* is the latter component of one of the evidence features, say e_1. (The initial component of e_1 maps from states to programs.) There are likely to be other evidence features as well. For example, there may be an evidence feature that determines whether all the occupants are free to watch the program at the time it is on. Once all the evidence and result features have been determined, the function

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

that is learned by Alkemy would be used along with the utility to give a rational policy to select an appropriate action (to recommend or not recommend any particular program.)

Note that ϵ_1 is the type *Density* Ω. This means that the hypothesis language used to learn *transition* should take account of properties of the density. In such a simple case as a density on the booleans, there are not many possibilities; different thresholds on the probability of $\top$ could be tried, for example.

6 Conclusion

This paper has shown how to integrate logical and probabilistic beliefs in modal higher-order logic. The key point is that the expressive power of the logic allows densities and other probabilistic concepts to appear in beliefs. Our approach is based on the standard axiomatic method that uses theories to model situations, and reasoning methods, such as theorem-proving and equational reasoning, to determine the values of terms (in the intended interpretation). The integration of logic and probability does not force any restriction on the logic employed; indeed, it is the expressive power of the logic that makes the integration actually possible. In fact, the logic we employ is considerably more expressive than those used in all other approaches to integration that we are aware of. Reasoning about probabilistic beliefs is realised through the Bach programming language that has special implementational support for this. In particular, there is support for operations, such as marginalisation, on large product densities. Also the standard compositional operations on densities can be neatly encoded in Bach. Beliefs can be acquired with the Alkemy learning system.

Future work includes completing the implementations of Bach and Alkemy, and applying the technology in challenging application areas, such as cognitive robotics and vision.

Acknowledgments

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

References

1. Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotics. MIT Press (2005)
2. Muggleton, S.: Stochastic logic programs. In De Raedt, L., ed.: Advances in Inductive Logic Programming, IOS Press (1996) 254–264
3. Milch, B., Marthi, B., Russell, S., Sontag, D., Ong, D.L., Kolobov, A.: BLOG: Probabilistic models with unknown objects. In Kaelbling, L., Saffiotti, A., eds.: Proceedings of the 19th International Joint Conference on Artificial Intelligence. (2005) 1352–1359
4. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning **62** (2006) 107–136

5. Milch, B., Russell, S.: First-order probabilistic languages: Into the unknown. In Muggleton, S., Otero, R., Tamaddoni-Nezhad, A., eds.: Proceedings of the 16th International Conference on Inductive Logic Programming. (2007)
6. Kersting, K., De Raedt, L.: Bayesian logic programming: Theory and tool. In Getoor, L., Taskar, B., eds.: Introduction to Statistical Relational Learning. MIT Press (2007)
7. Nilsson, N.J.: Probabilistic logic. Artificial Intelligence **28** (1986) 71–88
8. Halpern, J.: An analysis of first-order logics of probability. Artificial Intelligence **46** (1989) 311–350
9. Williamson, J.: Probability logic. In Gabbay, D., Johnson, R., Ohlbach, H., Woods, J., eds.: Handbook of the Logic of Inference and Argument: The Turn Toward the Practical. Volume 1 of Studies in Logic and Practical Reasoning. Elsevier (2002) 397–424
10. De Raedt, L., Kersting, K.: Probabilistic logic learning. SIGKDD Explorations **5** (2003) 31–48
11. Lloyd, J.: Knowledge representation and reasoning in modal higher-order logic. submitted for publication. `http://csl.anu.edu.au/~jwl` (2006)
12. 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
13. Lloyd, J., Ng, K.S.: Reflections on agent beliefs. In: Proceedings of the Declarative Agent Languages and Technologies Workshop (DALT 2007). (2007)
14. Lloyd, J., Ng, K.S., Veness, J.: Modal functional logic programming. Available at `http://rsise.anu.edu.au/~kee` (2007)
15. Friedman, N., Getoor, L., Koller, D., Pfeffer, A.: Learning probabilistic relational models. In: Proceedings of the 16th International Joint Conference on Artificial Intelligence. (1999) 1300–1307
16. 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
17. Rivest, R.: Learning decision lists. Machine Learning **2** (1987) 229–246
18. Eiter, T., Ibaraki, T., Makino, K.: Decision lists and related boolean functions. Theoretical Computer Science **270(1-2)** (2002) 493–524
19. Lloyd, J.: Logic for Learning. Cognitive Technologies. Springer (2003)
20. Ng, K.S.: Learning Comprehensible Theories from Structured Data. PhD thesis, Computer Sciences Laboratory, The Australian National University (2005)
21. Buntine, W.L.: A Theory of Learning Classification Rules. PhD thesis, School of Computing Science, University of Technology, Sydney (1992)
22. 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