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@cecs.anu.edu.au
[2] National ICT Australia
keesiong.ng@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 that is currently attracting substantial interest from researchers in artificial intelligence [2–7]. 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 [8–11]. However, the following quote from [10], 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 [11].

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 $\mathfrak{B}$ the Herbrand base. We can identify $\mathfrak{I}$ with the product space $\{0,1\}^{\mathfrak{B}}$ 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.

The next section gives a brief account of the logic we employ. In Section 3, the definition of a density and some of its properties are presented. Section 4 presents our approach to integrating logic and probability. Section 5 considers the idea that beliefs should be function definitions. Section 6 gives an extended example to illustrate the ideas. Section 7 gives some conclusions and future research directions.

2 Logic

We outline the most relevant aspects of the logic, focussing to begin with on the monomorphic version. We define types and terms, and give an introduction to the modalities that will be most useful in this paper. Full details of the logic, including its reasoning capabilities, can be found in [12].

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

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

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

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

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

Int is the type of the integers and *Real* is the type of the reals. $(List\ \sigma)$ is the type of lists whose items have type σ. Also $\sigma \rightarrow \Omega$ is the type of sets whose elements have type σ, since sets are identified with predicates; $\{\sigma\}$ is a synonym for $\sigma \rightarrow \Omega$ used when we are intuitively thinking of a term as a set of elements rather than as a predicate.

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

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

The intended meaning of $=_\alpha$ is identity (that is, $=_\alpha x\ y$ is $\top$ iff x and y are identical), the intended meaning of $\top$ is true, the intended meaning of $\bot$ is false, and the intended meanings of the connectives $\neg$, $\wedge$, $\vee$, $\longrightarrow$, $\longleftarrow$, and $\longleftrightarrow$ are as usual. The intended meanings of Σ_α and Π_α are that Σ_α maps a predicate to $\top$ iff the predicate maps at least one element to $\top$ and Π_α maps a predicate to $\top$ iff the predicate maps all elements to $\top$.

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

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

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

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

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

An important feature of higher-order logic is that it admits functions that can take functions as arguments and return functions as results. (First-order logic does not admit these so-called higher-order functions.) This fact can be exploited in applications, through the use of predicates to represent sets and densities to model uncertainty, for example.

As is well known, modalities can have a variety of meanings, depending on the application. Some of these are indicated here; much more detail can be found in [13], [14] and [12], for example.

In multi-agent applications, one meaning for $\Box_i \varphi$ is that 'agent i knows φ'. In this case, the modality $\Box_i$ is written as $\boldsymbol{K}_i$. A weaker notion is that of belief. In this case, $\Box_i \varphi$ means that 'agent i believes φ' and the modality $\Box_i$ is written as $\boldsymbol{B}_i$.

The modalities also have a variety of temporal readings. We will make use of the (past) temporal modalities $\bullet$ ('last') and $\blacksquare$ ('always in the past').

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 $\boldsymbol{\Box}$ denotes a sequence of modalities.

Composition is handled by the (reverse) composition function $\circ$ defined by $((f \circ g)\ x) = (g\ (f\ x))$.

The logic has a conventional possible-worlds semantics with higher-order interpretations at each world.

3 Densities

This section presents some standard notions of measure theory, particularly that of a density, which will be needed later [15].

Definition 4. *Let $(X, \mathcal{A}, \mu)$ be a measure space and $f : X \to \mathbb{R}$ a measurable function. Then f is a* density *(on $(X, \mathcal{A}, \mu)$) 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

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

If $(X, \mathcal{A}, \mu)$ is a measure space, then *Density* X denotes the set of densities on $(X, \mathcal{A}, \mu)$.

Some (higher-order) functions that operate on densities will be needed. The following two definitions give natural ways of 'composing' functions whose codomains are densities.

Definition 5. *Let* $(X, \mathcal{A}, \mu)$*,* $(Y, \mathcal{B}, \nu)$*, and* $(Z, \mathcal{C}, \xi)$ *be measure spaces. The function* $\natural : (X \to \mathit{Density}\ Y) \to (Y \to \mathit{Density}\ Z) \to (X \to \mathit{Density}\ Z)$ *is defined by*

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

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

Specialised to the discrete case, the definition is

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

Definition 6. *The function*

$$\S : \mathit{Density}\ Y \to (Y \to \mathit{Density}\ Z) \to \mathit{Density}\ Z$$

is defined by

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

where $f : \mathit{Density}\ Y$*,* $g : Y \to \mathit{Density}\ Z$*, and* $z \in Z$*.*

Specialised to the discrete case, the definition is

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

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

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

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

4 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 let *Density* σ denote the type of densities whose arguments have type σ. Any term of type *Density* σ, for some σ, is called a *density*. We also make available the functions from Section 3 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 $f : \sigma \rightarrow \tau$ for which there is some uncertainty about its values that we want to model. We do this with a function

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

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 *Density* τ.

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.

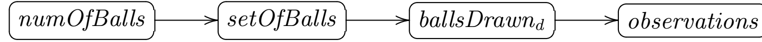
As well as representing knowledge, it is necessary to reason with it. We employ a declarative programming language called Bach for this purpose. Bach is a probabilistic modal functional logic programming language. Programs in the language are equational theories in modal higher-order logic. The reasoning system for the logic underlying Bach combines a theorem prover and an equational reasoning system [12, 16]. 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 is an example 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.

$$colour : Colour \rightarrow \Omega$$
$$(colour\ x) = (x = Blue) \vee (x = Green)$$

$$numOfBalls : Density\ Int$$
$$(numOfBalls\ x) = (poisson\ 6\ x)$$

$$poisson : Int \rightarrow Density\ Int$$
$$(poisson\ x\ y) = e^{-x} x^y / y!$$

$$setOfBalls : Int \rightarrow Density\ \{Ball\}$$
$$(setOfBalls\ n\ s) = if\ \exists x_1 \cdots \exists x_n.((colour\ x_1) \wedge \cdots \wedge (colour\ x_n) \wedge (s = \{(1, x_1), \ldots, (n, x_n)\})\ then\ 0.5^n\ else\ 0$$

$$ballsDrawn : Int \rightarrow \{Ball\} \rightarrow Density\ (List\ Ball)$$
$$(ballsDrawn\ d\ s\ x) =$$
$$if\ \exists x_1 \cdots \exists x_d.((s\ x_1) \wedge \cdots \wedge (s\ x_d) \wedge (x = [x_1, \ldots, x_d]))\ then\ (card\ s)^{-d}\ else\ 0$$

$$observations : (List\ Ball) \rightarrow Density\ (List\ Colour)$$
$$(observations\ x\ y) = if\ (length\ x) = (length\ y)\ then\ (obsProb\ x\ y)\ else\ 0$$

$$obsProb : (List\ Ball) \rightarrow (List\ Colour) \rightarrow Real$$
$$(obsProb\ []\ []) = 1$$
$$(obsProb\ (\#\ (x_1, y_1)\ z_1)\ (\#\ y_2\ z_2)) = (if\ (y_1 = y_2)\ then\ 0.8\ else\ 0.2) \cdot (obsProb\ z_1\ z_2)$$

$$
\begin{aligned}
&\mathit{joint} : \mathit{Int} \rightarrow \mathit{Density}\ (\mathit{Int} \times \{\mathit{Ball}\} \times (\mathit{List\ Ball}) \times (\mathit{List\ Colour}))\\
&(\mathit{joint}\ d\ (n, s, x, y)) =\\
&\qquad (\mathit{numOfBalls}\ n) \cdot (\mathit{setOfBalls}\ n\ s) \cdot (\mathit{ballsDrawn}\ d\ s\ x) \cdot (\mathit{observations}\ x\ y)
\end{aligned}
$$

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:

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

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

At this stage, it is interesting 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. In contrast, our view is that 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 proof-theoretic approach of building a theory (that has the intended interpretation as a model) and proving theorems with this theory instead. Here we adopt the well-established 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 [12]. 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.

5 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 7. *A* belief *is the definition of a function $f : \sigma \rightarrow \tau$ 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 $Density\ \nu$, 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 $\bullet\bullet$.)

To motivate Definition 7, we now consider a probabilistic extension of the rational agent architecture described in [17].

For this purpose, let S be the set of states of the agent and A the set of actions that the agent can apply. The set S is the underlying set of a measure space that has a σ-algebra $\mathcal{A}$ and a measure μ on $\mathcal{A}$. Often $(S, \mathcal{A}, \mu)$ is discrete so that $\mathcal{A}$ is the powerset of S and μ is the counting measure. The dynamics of the agent is captured by a function

$$transition : A \rightarrow S \rightarrow Density\ S.$$

In the discrete case, $transition(a)(s)(s')$ is the conditional probability that, given the state is s and action a is applied, there will be a transition to state s'.

Various specific rationality principles could be used; a widely used one is the principle of maximum expected utility [18, p.585] (namely, a rational agent should choose an action that maximises the agent's expected utility, where the utility is a real-valued function on the set of states). If this principle is used, it is assumed that the utility function is known to the agent and that the maximum expected value of the utility corresponds closely to the external performance measure. Under the principle of maximum expected utility, the policy function

$$policy : Density\ S \to A$$

is defined by

$$policy(\boldsymbol{s}) = \underset{a \in A}{\operatorname{argmax}}\ \mathbb{E}_{\boldsymbol{s}\ \S\ transition(a)}(utility),$$

for each $\boldsymbol{s} \in Density\ S$. Here $\mathbb{E}_{\boldsymbol{s}\ \S\ transition(a)}(utility)$ denotes the expectation of the random variable $utility : S \to \mathbb{R}$ with respect to the density $\boldsymbol{s}\ \S\ transition(a)$ (where $\S$ was defined in Section 3). If the current state density is $\boldsymbol{s}$, then the action selected is thus the one given by $policy(\boldsymbol{s})$.

In many complex applications, the agent is not given the definition of the function *transition* (or cannot be given this definition because it is impractical to specify it precisely enough); in such cases, the agent essentially has to learn the transition function from its experience in the environment. A common complication then is that the number of states may be very large, so large in fact that it is quite impractical to attempt to learn *directly* the definition of the function *transition*. Instead, an obvious idea is to partition the set of states into a much smaller number of subsets of states such that the states in each subset can be treated uniformly and learn a transition function over equivalence classes of states. We now explore this idea, showing how the expressivity of the logic can be exploited to turn this idea into a practical method that assists in the construction of agents.

It will be important to employ two different ways of partitioning the states. For this reason, two collections of functions on states are introduced. These are

$$e_i : S \to V_i,$$

where $i = 1, \ldots, n$, and

$$r_j : S \to W_j,$$

where $j = 1, \ldots, m$. Each function e_i, for $i = 1, \ldots, n$, is called an *evidence feature* and each function r_j, for $j = 1, \ldots, m$, is called a *result feature*. Each e_i and r_j is a feature that picks out a specific property of a state that is relevant to selecting actions. Evidence features are so-called because they pick out properties of the state that suggest the action that ought to be selected. Result features are so-called because they pick out properties of the state which result from applying

an action that can be usefully employed in the calculation of its utility. An interesting fact that emerges from applying these ideas to practical applications is how different the evidence features are compared with the result features [17].

Typically, the cardinality of the product spaces $V_1 \times \cdots \times V_n$ and $W_1 \times \cdots \times W_m$ are much smaller than the cardinality of S. Often, some V_i and W_j are simply the set of booleans. In an application that is at least partly continuous, they could be $\mathbb{R}$. There are two key functions associated with this construction. The function

$$(e_1, \dots, e_n) : S \to V_1 \times \cdots \times V_n$$

is defined by

$$(e_1, \dots, e_n)(s) = (e_1(s), \dots, e_n(s)),$$

for each $s \in S$. The space $V_1 \times \cdots \times V_n$ is assumed to have a suitable σ-algebra of measurable sets on it, so that $(e_1, \dots, e_n)$ is a measurable function. If μ is the measure on S, then $(e_1, \dots, e_n)^{-1} \circ \mu$ is the measure imposed on $V_1 \times \cdots \times V_n$. (Note that $(f \circ g)(x)$ means $g(f(x))$.) Similarly, the function

$$(r_1, \dots, r_m) : S \to W_1 \times \cdots \times W_m$$

is defined by

$$(r_1, \dots, r_m)(s) = (r_1(s), \dots, r_m(s)),$$

for each $s \in S$. The space $W_1 \times \cdots \times W_m$ is also assumed to have a suitable σ-algebra of measurable sets on it, so that $(r_1, \dots, r_m)$ is a measurable function. Similarly, $(r_1, \dots, r_m)^{-1} \circ \mu$ is the measure imposed on $W_1 \times \cdots \times W_m$.

Now, instead of the transition function

$$\mathit{transition} : A \to S \to \mathit{Density}\ S,$$

one works with a function

$$\mathit{transition}' : A \to V_1 \times \cdots \times V_n \to \mathit{Density}\ W_1 \times \cdots \times W_m.$$

The motivation for doing this is that *transition*$'$ should be more amenable to being learned because of the much smaller cardinalities of its domain and codomain. Some precision is lost in working with $V_1 \times \cdots \times V_n$ and $W_1 \times \cdots \times W_m$ instead of S in this way, as discussed below; to make up for this, informative choices of the evidence and result features need to be made. The policy is now defined by

$$\mathit{policy}(\boldsymbol{s}) = \operatorname*{argmax}_{a \in A}\ \mathbb{E}_{\boldsymbol{s}\ \S\ ((e_1,\dots,e_n) \circ \mathit{transition}'(a))}(\mathit{utility}'),$$

for each $\boldsymbol{s} \in \mathit{Density}\ S$. Here

$$\mathit{utility}' : W_1 \times \cdots \times W_m \to \mathbb{R}.$$

The result features are intended to be chosen so that $(r_1, \ldots, r_m) \circ \mathit{utility}'$ gives the utility of each state. Note that

$$(e_1, \ldots, e_n) \circ \mathit{transition}'(a) : S \rightarrow \mathit{Density}\ W_1 \times \cdots \times W_m,$$

so that $\boldsymbol{s}\ \S\ ((e_1, \ldots, e_n) \circ \mathit{transition}'(a))$ is a density on $W_1 \times \cdots \times W_m$.

If the action selected by the policy is a_{max}, then the state density that results by applying this action is

$$\boldsymbol{s}\ \S\ ((e_1, \ldots, e_n) \circ \mathit{transition}'(a_{max}) \circ \overline{(r_1, \ldots, r_m)}).$$

Here, since $(r_1, \ldots, r_m) : S \rightarrow W_1 \times \cdots \times W_m$, it follows from [15, Theorem 4.1.11] that one can define the function

$$\overline{(r_1, \ldots, r_m)} : \mathit{Density}\ W_1 \times \cdots \times W_m \rightarrow \mathit{Density}\ S$$

by

$$\overline{(r_1, \ldots, r_m)}(h) = (r_1, \ldots, r_m) \circ h,$$

for each $h \in \mathit{Density}\ W_1 \times \cdots \times W_m$. Thus

$$(e_1, \ldots, e_n) \circ \mathit{transition}'(a_{max}) \circ \overline{(r_1, \ldots, r_m)} : S \rightarrow \mathit{Density}\ S$$

and so

$$\boldsymbol{s}\ \S\ ((e_1, \ldots, e_n) \circ \mathit{transition}'(a_{max}) \circ \overline{(r_1, \ldots, r_m)}) : \mathit{Density}\ S,$$

as required.

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 7.

At this point, the advantages and disadvantages of introducing features are clearer. The main advantage is that they can make learning the transition function feasible when otherwise it wouldn't be because of the vast size of the state space. The main disadvantage is that some precision in the update of the state density during the agent cycle is lost since this is now mediated by passing through $V_1 \times \cdots \times V_n$ and $W_1 \times \cdots \times W_m$. This shows the crucial importance of finding suitable features; if these can be found, and experience has shown that this can be hard for some applications, then the loss of precision is likely to be small and there is everything to gain. Choosing the 'right' (general form of) features is generally a problem that has to be solved before deployment by the designer of the agent, although some techniques are known that allow agents to discover such information autonomously. For example, it is likely that the features will need to capture beliefs about the beliefs of other agents, beliefs about

temporal aspects of the environment, and will have to cope with uncertainty. The logic of this paper is ideal for the representation of such properties. Agents can learn the precise definition of a feature (whose general form is given by an hypothesis language) during deployment by machine learning techniques. This eases the task of the designer since only the general forms of the features need specifying before deployment. In a dynamic environment, this adaptive capability of an agent is essential, of course.

The above description generalises the agent architecture presented in [17] 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 that beliefs can take in more detail. Some beliefs can be specified directly by the designer 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 3 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.

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

where $\Box$ is a (possibly empty) sequence of modalities, $p_1, \ldots, p_n$ are predicates that can be modal and/or higher order, and $v_0, v_1, \ldots, v_n$ are suitable values. Such a belief is a definition for the function f in the context of the modal sequence $\Box$. 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 [19, 20]. Also this decision-list form of beliefs is highly convenient for acquisition using some kind of learning algorithm [21–24]. Towards that end, the Alkemy machine learning system [22, 23] is being extended with the ability to acquire modal and probabilistic beliefs.

6 Illustration

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

Example 2. 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 $likes : Program \rightarrow Density\ \Omega$. 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:

$$\boldsymbol{B}_m\boldsymbol{B}_a\ \forall x.((likes\ x) =\ \varphi) \tag{2}$$
$$\boldsymbol{B}_m\boldsymbol{B}_b\ \forall x.((likes\ x) =\ \psi) \tag{3}$$
$$\boldsymbol{B}_m\boldsymbol{B}_c\ \forall x.((likes\ x) =\ \xi) \tag{4}$$

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 [25].

Figure 1 is a typical definition acquired incrementally using real data. In the beginning, the belief base contains the formula

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

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 1. 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 much 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.

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

$$\boldsymbol{B}_m\, \boldsymbol{B}_a\, \forall x.((likes\ x) =$$
$$\quad if\ (projTitle \circ (=\ \text{“NFL Football”})\ x)\ then\ \lambda y.if\ (y = \top)\ then\ 1\ else\ if\ (y = \bot)\ then\ 0\ else\ 0$$
$$\quad else\ if\ (projTitle \circ (existsWord\ (=\ \text{“sport”}))\ x)\ then\ \lambda y.if\ (y = \top)\ then\ 0.7$$
$$\qquad else\ if\ (y = \bot)\ then\ 0.3\ else\ 0$$
$$\quad else\ (\bullet likes\ x))$$

$$\bullet \boldsymbol{B}_m\, \boldsymbol{B}_a\, \forall x.((likes\ x) =$$
$$\quad if\ (projGenre \circ (=\ Documentary)\ x)\ then\ \lambda y.if\ (y = \top)\ then\ 0.9\ else\ if\ (y = \bot)\ then\ 0.1\ else\ 0$$
$$\quad else\ if\ (projGenre \circ (=\ Movie)\ x)\ then\ \lambda y.if\ (y = \top)\ then\ 0.75\ else\ if\ (y = \bot)\ then\ 0.25\ else\ 0$$
$$\quad else\ (\bullet likes\ x))$$

$$\bullet^2 \boldsymbol{B}_m\, \boldsymbol{B}_a\, \forall x.((likes\ x) =$$
$$\quad if\ (projGenre \circ (=\ Documentary)\ x)\ then\ \lambda y.if\ (y = \top)\ then\ 1\ else\ if\ (y = \bot)\ then\ 0\ else\ 0$$
$$\quad else\ if\ (projGenre \circ (=\ Drama)\ x)\ then\ \lambda y.if\ (y = \bot)\ then\ 0.8\ else\ if\ (y = \top)\ then\ 0.2\ else\ 0$$
$$\quad else\ (\bullet likes\ x))$$

$$\bullet^3 \boldsymbol{B}_m \blacksquare \boldsymbol{B}_a \forall x.((likes\ x) = \lambda y.if\ (y = \top)\ then\ 0.5\ else\ if\ (y = \bot)\ then\ 0.5\ else\ 0).$$

Fig. 1. Part of the belief base of the agent

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

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

$$transition' : A \to V_1 \times \cdots \times V_n \to Density\ W_1 \times \cdots \times W_m$$

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, for this example, elements in V_1 have 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.

7 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 well-established method that uses theories to model situations, and reasoning procedures, 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., 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. Shirazi, A., Amir, E.: Probabilistic modal logic. In Holte, R., Howe, A., eds.: Proceedings of the 22nd AAAI Conference on Artificial Intelligence. (2007) 489–495
8. Nilsson, N.: Probabilistic logic. Artificial Intelligence **28**(1) (1986) 71–88
9. Halpern, J.: An analysis of first-order logics of probability. Artificial Intelligence **46** (1989) 311–350
10. 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
11. De Raedt, L., Kersting, K.: Probabilistic logic learning. SIGKDD Explorations **5**(1) (2003) 31–48
12. Lloyd, J.: Knowledge representation and reasoning in modal higher-order logic. `http://users.rsise.anu.edu.au/~jwl` (2007)
13. Fagin, R., Halpern, J., Moses, Y., Vardi, M.: Reasoning about Knowledge. MIT Press (1995)
14. Gabbay, D., Kurucz, A., Wolter, F., Zakharyaschev, M.: Many-Dimensional Modal Logics: Theory and Applications. Studies in Logic and The Foundations of Mathematics, Volume 148. Elsevier (2003)
15. Dudley, R.: Real Analysis and Probability. Cambridge University Press (2002)
16. Lloyd, J., Ng, K.S.: Reflections on agent beliefs. In Baldoni, M., Son, T., van Riemsdijk, M.B., Winikoff, M., eds.: Proceedings of the Workshop on Declarative Agent Languages and Technologies (DALT 2007). (2007) 99–114
17. 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
18. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Second edn. Prentice-Hall (2002)
19. Rivest, R.: Learning decision lists. Machine Learning **2**(3) (1987) 229–246
20. Eiter, T., Ibaraki, T., Makino, K.: Decision lists and related boolean functions. Theoretical Computer Science **270(1-2)** (2002) 493–524
21. 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
22. Lloyd, J.: Logic for Learning. Cognitive Technologies. Springer (2003)
23. Ng, K.S.: Learning Comprehensible Theories from Structured Data. PhD thesis, Computer Sciences Laboratory, The Australian National University (2005)
24. Buntine, W.L.: A Theory of Learning Classification Rules. PhD thesis, School of Computing Science, University of Technology, Sydney (1992)
25. 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