Available online at www.sciencedirect.com

ScienceDirect

Journal of Applied Logic 7 (2009) 218–238

JOURNAL OF APPLIED LOGIC

www.elsevier.com/locate/jal

Probabilistic reasoning in a classical logic

K.S. Ng[a,*], J.W. Lloyd[b]

[a] *Making Sense of Data Group, National ICT Australia*
[b] *College of Engineering and Computer Science, The Australian National University*

Received 1 July 2007; received in revised form 2 November 2007; accepted 10 November 2007

Available online 15 January 2008

Abstract

We offer a view on how probability is related to logic. Specifically, we argue against the widely held belief that standard classical logics have no direct way of modelling the certainty of assumptions in theories and no direct way of stating the certainty of theorems proved from these (uncertain) assumptions. The argument rests on the observation that probability densities, being functions, can be represented and reasoned with naturally and directly in (classical) higher-order logic.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Probabilistic reasoning; Classical logic; Higher-order logic; Integrating logic and probability

1. Introduction

The search for a tight integration of probability and logic is a problem that is currently attracting substantial interest [19,21,23,34,36,43,45]. 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 [12, 20,39,49]. However, the following quote from Williamson [49] 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".

We show how probability can be handled in higher-order logic in this paper and, in doing so, offer a counter-argument to the widely held belief that standard classical logics have no direct way of modelling the certainty of assumptions in theories and no direct way of stating the certainty of theorems proved from these (uncertain) assumptions. 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 the literature. A more extensive survey of this and other approaches can be found in [35].

The standard logical setting adopted for integrating logic and probability 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

* Corresponding author.
E-mail addresses: keesiong.ng@nicta.com.au (K.S. Ng), john.lloyd@anu.edu.au (J.W. Lloyd).

1570-8683/$ – see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.jal.2007.11.008

observable). The environment is modelled as a probability distribution over the collection of first-order models (over some suitable alphabet for the application at hand). The intuition is that any of these models could be the actual environment but that some models are more likely than others and this information is given by the distribution on the models. 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 $Pr(\varphi \mid \psi)$?

We now formalise this situation. Let $\mathcal{I}$ be the set of interpretations and p a probability measure on the σ-algebra of all subsets of this set. Define the random variable $X_\varphi : \mathcal{I} \to \mathbb{R}$ 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 $Pr(\varphi \mid \psi)$ can be written in the form

$$p(X_\varphi = 1 \mid X_\psi = 1)$$

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 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 $\mathcal{I}$ denote the (finite) set of Herbrand interpretations and $\mathcal{B}$ the Herbrand base. We can identify $\mathcal{I}$ 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, the distribution can be represented either directly or more compactly using formalisms like Bayesian networks or Markov random fields. In [43], 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 [23], conditional probability distributions are attached to clauses to give a Bayesian network. Closely related to [23] is [41], in which probability distributions are attached to sets of literals capturing alternative scenarios and logic programming is used to generate a distribution on possible worlds. In [4], A-Prolog [18] is extended with so-called probabilistic atoms to generate distributions on possible worlds using answer set programming. In [34], 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 more expressive logic, higher-order logic. Also, in our approach, the theory plays a central role and probabilistic reasoning all takes place in the context of the theory. The key idea is simply to allow probability density functions and operations on them to appear explicitly in theories. This is possible because higher-order logic is a function-based language, and densities are just a special kind of function. As we shall see, the higher-orderness of the logic will also be essential to achieve the desired integration of logic and probability.

The main contributions of this paper can be summarised as follows.

– We show how higher-order logic can be used to naturally represent probabilistic concepts.
– We outline a reasoning system (essentially a functional logic programming language) for reasoning with theories that include probabilistic concepts.
– We illustrate the ideas with a wide variety of examples, most of which have been studied by other authors and therefore allow a comparison of existing approaches with that of this paper.

The next section gives some mathematical preliminaries. The reasoning system is then described in Section 3. Examples illustrating our approach to reasoning with uncertainty are presented in Sections 4 and 5. A general discussion then follows in Section 6.

2. Mathematical preliminaries

We review a formulation of higher-order logic based on Church's simple theory of types [9] in Section 2.1. (More complete accounts of the logic can be found in [28] and [29]. Other references on higher-order logic include [3,26, 47,48], and [46].) We then introduce a modicum of measure theory in Section 2.2 and show how probability density functions can be naturally represented in the logic.

2.1. Logic

Definition 1. An *alphabet* consists of three sets: a set $\mathfrak{T}$ of type constructors; a set $\mathfrak{C}$ of constants; and 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 $x, y, z, \ldots$.

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 $\alpha \to \beta$ is a type.
3. If $\alpha_1, \ldots, \alpha_n$ are types, then $\alpha_1 \times \cdots \times \alpha_n$ is a type.

We use the convention that $\to$ is right associative. So when we write $\alpha \to \beta \to \gamma \to \kappa$, we mean $\alpha \to (\beta \to (\gamma \to \kappa))$.

Besides Ω, here are some other common types we will need.

Example 1. The type of the integers is denoted by *Int*, and the type of the reals by *Real*. Also $(List\ \sigma)$ is the type of lists whose elements have type σ. Here *Int*, *Real* and *List* are all type constructors. The first two have arity 0 and the last has arity 1.

Example 2. A function that maps elements of type α to elements of type β has type $\alpha \to \beta$. Since sets are identified with predicates in the logic (see Example 5 below), sets whose elements have type σ have type $\sigma \to \Omega$. We sometimes write $\{\sigma\}$ as a synonym for $\sigma \to \Omega$ when we want to make a distinction between sets and predicates.

The set $\mathfrak{C}$ always includes the following constants: $\top$ and $\bot$ having signature Ω; $=_\alpha$ having signature $\alpha \to \alpha \to \Omega$ for each type α; $\neg$ having signature $\Omega \to \Omega$; $\wedge$, $\vee$, $\longrightarrow$, and $\longleftarrow$ having signature $\Omega \to \Omega \to \Omega$; and Σ_α and Π_α having signature $(\alpha \to \Omega) \to \Omega$ for each type α. The intended meaning of $\top$ is true, and that of $\bot$ is false. The intended meaning of $=_\alpha$ is identity (that is, $=_\alpha x\ y$ is $\top$ iff x and y are identical), and the intended meanings of the connectives $\neg$, $\wedge$, $\vee$, $\longrightarrow$ and $\longleftarrow$ 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 $\top$.

Other useful constants we will usually have in applications include the integers, the real numbers, the characters, and data constructors like $\#_\sigma : \sigma \to (List\ \sigma) \to (List\ \sigma)$ and $[]_\sigma : List\ \sigma$ for constructing lists where the elements have type σ. The notation $C : \sigma$ is used to denote that the constant C has signature σ.

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. (Abstraction) 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. (Application) 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. (Tuple) 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 $\alpha_1 \times \cdots \times \alpha_n$.

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).

Example 3. Constants like $\top$, 42, 3.11, and $+$ with signature $Int \to Int \to Int$ are terms. Variables like x, y, z are terms. An example of a term that can be formed using abstraction is $\lambda x.((+\ x)\ x)$, whose intended meaning is a function that takes a number x and returns $x + x$. To apply that function to the constant 42, for example, we use application to form the term $(\lambda x.((+\ x)\ x)\ 42)$.

Example 4. The term $(\#_{Int}\ 2\ (\#_{Int}\ 3\ []_{Int}))$ represents a list with the numbers 2 and 3 in it, obtained via a series of applications from the constants $\#_{Int}$, $[]_{Int}$, 2, and 3, each of which is a term. For convenience, we sometimes write $[2, 3]$ to represent the same list.

Example 5. Sets are identified with predicates in the logic. Thus, the term

$$\lambda x.((\vee\ ((=_{Int}\ x)\ 2))\ ((=_{Int}\ x)\ 3)) \tag{1}$$

can be used to represent a set with the integers 2 and 3 in it. We often use infix notation for common function symbols like equality and the connectives. We also adopt the convention that applications are left-associative; thus, $(f\ x\ y)$ means $((f\ x)\ y)$. The above conventions allow us to write $\lambda x.((x =_{Int} 2) \vee (x =_{Int} 3))$ to mean (1) above. For convenience, we sometimes also write $\{2, 3\}$ to represent the same set. Since sets are predicates, set membership test is obtained using function application. Let s denote (1) above. To check whether a number y is in the set, we write $(s\ y)$.

The polymorphic version of the logic extends what is given above by also having available type variables (denoted by $a, b, c, \ldots$). The definition of a type as above is then extended to polymorphic types that may contain type variables 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 constants like $\exists_\alpha$, $\forall_\alpha$, $=_\alpha$, $[]_\alpha$ and $\#_\alpha$, since the types associated with these are now inferred from the context. The universal closure of a formula φ is denoted by $\forall(\varphi)$.

The logic can be given a rather conventional Henkin semantics. For each type σ, $\mathcal{D}_\sigma$ denotes the domain of σ in the semantics of the logic.

2.2. Densities

Probability distributions can be described formally using measure theory. We will assume familiarity with the basic concepts of measure theory in the following. Readers are referred to [5,14] for more details. Suppose an experiment is performed that randomly returns some element of a set Y (according to some distribution). This is modelled probabilistically by supposing that there is a σ-algebra $\mathcal{A}$ of measurable subsets (that is, events) of Y and a probability measure ν on $\mathcal{A}$ giving the probability space $(Y, \mathcal{A}, \nu)$. Then the probability that a randomly chosen element (that is, sample point) lies in a subset A of Y is given by $\nu(A)$.

Probability measures are not convenient to deal with in many applications (partly because the σ-algebra that is the domain of the probability would have to be made explicit); instead, it is easier to work with densities of probability measures. To set this up, suppose there is an underlying σ-finite measure μ on $\mathcal{A}$. Under a weak condition (the absolute continuity of ν with respect to μ), the Radon–Nikodym theorem [14, Theorem 5.5.4] states that there exists a non-negative integrable function h such that

$$\nu(A) = \int_A h\, \mathrm{d}\mu,$$

for all $A \in \mathcal{A}$. Conversely, given such an h, the preceding equation defines a finite measure on $\mathcal{A}$ that is absolutely continuous with respect to μ.

There are two cases of particular interest for the underlying measure μ. In the first case, Y is a countable set and μ is the counting measure; this is the discrete case. In the second case, Y is $\mathbb{R}^n$ ($n \geqslant 1$) and μ is Lebesgue measure; this is the continuous case.

A density is a non-negative, integrable function h on a set Y such that $\int_Y h \, d\mu = 1$. In effect, in all the situations that are likely to be met in practice, the Radon–Nikodym theorem shows that one can work with densities instead of probability measures.

To capture the above notions in our logical setting, we introduce in the logic the type synonym *Density* $a \equiv a \rightarrow$ *Real*. (Here a is a parameter, that is, type variable; so *Density* a is a polymorphic type.) The intended meaning of a term of type *Density* τ in the semantics is a density over $\mathcal{D}_\tau$, not some arbitrary real-valued function. Any term of type *Density* τ, for some τ, is called a *density*. We will see some common operations for manipulating densities in Section 3.

3. A reasoning system

We next define a suitable reasoning system. Before doing that, we give some remarks on the expressiveness of the logic and how that can be controlled, noting that although higher-order logic is already widely used in many areas of computer science, there remain some reservations about its use in AI applications, which is manifested in the widespread belief that higher-order logic is *too* expressive and is thus computationally hard to control.

The language is indeed expressive. In higher-order logic, one can quantify over arbitrary domains, including over sets of functions. One can also have higher-order functions, i.e., functions that take functions as arguments and/or return functions as values. Both these facilities, which are extremely useful in practice and serve important roles in our approach to integrating logic and probability, are not readily available in first-order logic.

The assumption that higher-order logic is, by virtue of its expressiveness, necessarily hard to control computationally is, however, not true. There are some problematic operations in higher-order logic: unification of terms, for example, is undecidable. But the point to note is simply that one can do a lot in the logic without ever resorting to these operations. To illustrate this point, observe that the reasoning system described in the following is a useful subset of higher-order logic that is both expressive and tractable, in the sense that it admits the usual higher-order constructs but is still efficient enough to be used as a programming language. The system uses the linear-time (one-way) matching of terms instead of the difficult (two-way) unification of terms for pattern matching. Also it captures a significant part of theorem-proving via a mechanism for doing equational reasoning.

The reasoning system described next is a subset of a more general system called Bach [30]. This subset is a computation system that significantly extends existing functional programming languages by adding logic programming facilities. For convenience, we will call this subsystem Bach in this paper. Bach is closely related to Haskell, being a strict superset. Haskell allows pattern matching only on data constructors. Bach extends this by allowing pattern matching on function symbols and lambda abstractions in addition to data constructors. Bach also allows reduction of terms inside lambda abstractions, an operation not permitted in Haskell. There is also a big overlap between Bach and Prolog: any pure Prolog program can be mechanically translated into Bach using Clark's completion [10].

The inference mechanism underlying Bach is given in Definition 4 below. We first establish some terminology. The occurrence o of a subterm s in a term t is a description of the path from the root of t to s. The notation $t[s/r]_o$ denotes the term obtained from t by replacing s at occurrence o with r. The notation $t\{x/r\}$ denotes the term obtained from t by replacing every free occurrence of variable x in t with r. Two terms are α-equivalent if they are identical up to a change of bound variable names.

Definition 4. Let $\mathcal{T}$ be a theory. A *computation with respect to* $\mathcal{T}$ is a sequence $\{t_i\}_{i=1}^n$ of terms such that for $i = 1, \ldots, n-1$, there is a subterm s_i of t_i at occurrence o_i, a formula $\forall(u_i = v_i)$ in $\mathcal{T}$, and a substitution θ_i such that $u_i\theta_i$ is α-equivalent to s_i and t_{i+1} is $t_i[s_i/v_i\theta_i]_{o_i}$.

The term t_1 is called the *goal* of the computation and t_n is called the *answer*. Each subterm s_i is called a *redex* (short for reducible expression). The formula $\forall(t_1 = t_n)$ is called the *result* of the computation.

Theorem 1. *The result of a computation with respect to* $\mathcal{T}$ *is a consequence of* $\mathcal{T}$.

Theorem 1 shows that computation as defined is essentially a form of theorem proving. The proof of Theorem 1 is rather involved and is omitted here; the proof of a more general case of the theorem can be found in [30, Sect. 6].

Computations generally require use of definitions of $=$, the connectives and quantifiers, and some other basic functions. These definitions constitute what we call the standard equality theory. The complete list of equations in its various versions can be found in [27,29,30]. Here we give some examples of equations we need.

$$(\mathit{if}\ \top\ \mathit{then}\ u\ \mathit{else}\ v) = u \tag{2}$$

$$(\mathit{if}\ \bot\ \mathit{then}\ u\ \mathit{else}\ v) = v \tag{3}$$

$$(\boldsymbol{w}\ (\mathit{if}\ \boldsymbol{x} = \boldsymbol{t}\ \mathit{then}\ u\ \mathit{else}\ v)) = (\mathit{if}\ \boldsymbol{x} = \boldsymbol{t}\ \mathit{then}\ (\boldsymbol{w}\{\boldsymbol{x}/\boldsymbol{t}\}\ u)\ \mathit{else}\ (\boldsymbol{w}\ v)) \quad \text{—— where } \boldsymbol{x} \text{ is a variable.} \tag{4}$$

$$((\mathit{if}\ \boldsymbol{x} = \boldsymbol{t}\ \mathit{then}\ u\ \mathit{else}\ v)\ \boldsymbol{w}) = (\mathit{if}\ \boldsymbol{x} = \boldsymbol{t}\ \mathit{then}\ (u\ \boldsymbol{w}\{\boldsymbol{x}/\boldsymbol{t}\})\ \mathit{else}\ (v\ \boldsymbol{w})) \quad \text{—— where } \boldsymbol{x} \text{ is a variable.} \tag{5}$$

$$(\lambda x.\boldsymbol{u}\ t) = \boldsymbol{u}\{x/t\} \tag{6}$$

$$\exists x_1.\cdots\exists x_n.(\boldsymbol{x} \wedge (x_1 = \boldsymbol{u}) \wedge \boldsymbol{y}) = \exists x_2.\cdots\exists x_n.(\boldsymbol{x}\{x_1/\boldsymbol{u}\} \wedge \boldsymbol{y}\{x_1/\boldsymbol{u}\}) \quad \text{—— where } x_1 \text{ is not free in } \boldsymbol{u}. \tag{7}$$

$$\forall x_1.\cdots\forall x_n.(\boldsymbol{x} \wedge (x_1 = \boldsymbol{u}) \wedge \boldsymbol{y} \longrightarrow \boldsymbol{v}) = \forall x_2.\cdots\forall x_n.(\boldsymbol{x}\{x_1/\boldsymbol{u}\} \wedge \boldsymbol{y}\{x_1/\boldsymbol{u}\} \longrightarrow \boldsymbol{v}\{x_1/\boldsymbol{u}\}) \tag{8}$$

—— where x_1 is not free in $\boldsymbol{u}$.

The first two equations are standard. The other equations are schemas. A schema is intended to stand for the collection of formulas that can be obtained from the schema by replacing its syntactical variables (typeset in bold above) with terms that satisfy the side conditions, if there are any. The first four equations above are useful for simplifying *if_then_else* expressions. Eq. (6) provides β-reduction. Equations like (7) and (8) are used to provide logic programming idioms in the context of functional computations. Note that the *if_then_else* function has the signature $\Omega \times a \times a \to a$. A term of the form $\mathit{if}\ x\ \mathit{then}\ y\ \mathit{else}\ z$ is really syntactic sugar for $(\mathit{if\_then\_else}\ (x, y, z))$.

Example 6. We now look at an example computation. Consider the following definition of $f : \mathit{Char} \to \mathit{Int}$:

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

where $A, B, C : \mathit{Char}$. With such a definition, it is straightforward to compute in the 'forward' direction. Thus, for example, $(f\ B)$ can be computed in the obvious way to produce the answer 21. Less obviously, the definition can be used to compute in the 'reverse' direction. For example, Fig. 1 shows the computation of $\{x \mid (f\ x) = 42\}$, which produces the answer $\{A, C\}$. (The notation $\{x \mid t\}$ is syntactic sugar for the term $\lambda x.t$ when t has type Ω.) The computation makes essential use of Eqs. (4) and (5) from the standard equality theory.

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

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

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

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

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

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

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

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

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

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

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

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

Fig. 1. Computation of $\{x \mid (f\ x) = 42\}$. The redexes are underlined.

Some useful functions

In addition to the rules in the standard equality theory, we will find the functions discussed below useful for probabilistic reasoning.

We first introduce some functions for doing summations, a ubiquitous operation needed in the manipulation of discrete density functions. We will need a function *sum1* that computes $\sum_x f(x)$, where f is a real-valued function having finite support (that is, takes a non-zero value on at most finitely many elements of its domain) and x ranges over the domain of f.

$$\begin{aligned}
&\mathit{sum1} : (a \to \mathit{Real}) \to \mathit{Real}\\
&(\mathit{sum1}\ \lambda x.\mathit{if}\ x = u\ \mathit{then}\ v\ \mathit{else}\ \boldsymbol{w}) = v + (\mathit{sum1}\ \lambda x.\boldsymbol{w})\\
&(\mathit{sum1}\ \lambda x.0) = 0
\end{aligned}$$

Here it is assumed that the argument to *sum1* has the syntactic form of an abstraction over a nested *if_then_else* for which there are no repeated occurrences of the tests $x = u$. If there are such repeats, it is easy enough to remove them [29, p.189]. Similar remarks apply to *sum2* below.

The function *sum2* is used to compute expressions of the form $\sum_{x \in S} f(x)$, where f is a real-valued function having finite support and S is a subset of the domain of f.

$$\begin{aligned}
&\mathit{sum2} : \{a\} \to (a \to \mathit{Real}) \to \mathit{Real}\\
&(\mathit{sum2}\ s\ \lambda x.\mathit{if}\ x = u\ \mathit{then}\ v\ \mathit{else}\ \boldsymbol{w}) = (\mathit{if}\ (s\ u)\ \mathit{then}\ v\ \mathit{else}\ 0) + (\mathit{sum2}\ s\ \lambda x.\boldsymbol{w})\\
&(\mathit{sum2}\ s\ \lambda x.0) = 0
\end{aligned}$$

Note that *sum1* and *sum2* are closely related in that $(\mathit{sum2}\ \lambda y.\top\ f) = (\mathit{sum1}\ f)$.

The last of our summation functions is a variant of *sum2* where we exploit the fact that the S in $\sum_{x \in S} f(x)$ can be enumerated in the form of either a nested *if_then_else* or a nested disjunction.

$$\begin{aligned}
&\mathit{sum3} : \{a\} \to (a \to \mathit{Real}) \to \mathit{Real}\\
&(\mathit{sum3}\ \lambda x.\mathit{if}\ (x = y)\ \mathit{then}\ \top\ \mathit{else}\ \boldsymbol{w}\ f) = (f\ y) + (\mathit{sum3}\ \lambda x.\boldsymbol{w}\ f)\\
&(\mathit{sum3}\ \lambda x.(x = y)\ f) = (f\ y)\\
&(\mathit{sum3}\ \lambda x.((x = y) \vee \boldsymbol{w})\ f) = (f\ y) + (\mathit{sum3}\ \lambda x.\boldsymbol{w}\ f)\\
&(\mathit{sum3}\ \lambda x.(\boldsymbol{w} \vee (x = y))\ f) = (f\ y) + (\mathit{sum3}\ \lambda x.\boldsymbol{w}\ f)\\
&(\mathit{sum3}\ \lambda x.(\boldsymbol{v} \vee \boldsymbol{w})\ f) = (\mathit{sum3}\ \lambda x.\boldsymbol{v}\ f) + (\mathit{sum3}\ \lambda x.\boldsymbol{w}\ f)\\
&(\mathit{sum3}\ \lambda x.\bot\ f) = 0
\end{aligned}$$

The second set of functions below plays a crucial role in connecting probabilistic and non-probabilistic function definitions in reasoning. The indicator function $\mathbb{I} : \Omega \to \mathit{Real}$ is defined by $(\mathbb{I}\ \top) = 1$ and $(\mathbb{I}\ \bot) = 0$. Composition of functions is handled using

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

defined by $((f \circ g)\ x) = (g\ (f\ x))$. We also have various kinds of convolutions for composing densities. Two such functions are given here; more examples can be found in [32]. The mathematical definitions are stated first, followed by their definitions in the logic.

Definition 5. The function

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

is defined by

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

where $f : Density\ Y$, $g : Y \to Density\ Z$, and $z \in Z$. Specialised to the discrete case, the definition is

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

Definition 6. The function

$$\$: Density\ Y \to (Y \to Z) \to Density\ Z$$

is defined by

$$(f \ \$\ g)(z) = \int_{g^{-1}(z)} f(y)\, \mathrm{d}\nu(y),$$

where $f : Density\ Y$, $g : Y \to Z$, and $z \in Z$. Specialised to the discrete case, the definition is

$$(f \ \$\ g)(z) = \sum_{y \in g^{-1}(z)} f(y).$$

We define in the logic the discrete case of the two functions as follows:

$$\S : Density\ b \to (b \to Density\ c) \to Density\ c$$
$$(f \ \S\ g) = \lambda z.(sum1\ \lambda y.((f\ y) \times ((g\ y)\ z)))$$
$$\$: Density\ b \to (b \to c) \to Density\ c$$
$$(f \ \$\ g) = \lambda z.(sum2\ \lambda y.((g\ y) = z)\ f).$$

4. Logical reasoning with uncertain premises: An extended example

The key idea behind our approach to integrating logic and probability 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

$$f' : \sigma \to Density\ \tau,$$

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 $Density\ \tau$.

This simple idea turns out to be a powerful and convenient way of modelling uncertainty with logical theories in diverse applications. Note carefully the use that has been made of higher-order logic here. Functions whose values are densities are higher-order functions that can only be modelled by indirect and cumbersome ways with first-order logic.

We now give an extended example showing how an agent can logically reason with a mix of certain and uncertain premises expressed in theories. Consider an agent that makes recommendations of TV programs to a user. The agent has access to the TV guide through the definition of the function *tv_guide*. It also knows about the user's preferences for TV programs through the definition of the function *likes*, the uncertainty in which is modelled by densities in its codomain. Suppose now that the agent is asked to make a recommendation about a program at a particular occurrence (that is, date, time, and channel), except that there is some uncertainty in the actual occurrence intended by the user; this uncertainty, which is modelled in the definition of the density *choice*, could come about, for example, if the user asked the somewhat ambiguous question "Are there any good programs on *ABC* during dinner time?". The question the agent needs to answer is the following: given the uncertainty in *choice* and *likes*, what is the probability that the user likes the program that is on at the occurrence intended by the user.

This situation is modelled as follows. First, we give some type synonyms.

$$Occurrence = Date \times Time \times Channel$$
$$Date = Day \times Month \times Year$$

$$Time = Hour \times Minute$$

$$Program = Title \times Duration \times Genre \times Classification \times Synopsis.$$

Here is the definition of the density *choice* that models the uncertainty in the intended occurrence.

$$\begin{aligned}
&choice : Density\ Occurrence\\
&\forall x.((choice\ x) =\\
&\qquad if\ x = ((21, 7, 2007), (19, 30), ABC)\ then\ 0.8\\
&\qquad else\ if\ x = ((21, 7, 2007), (20, 30), ABC)\ then\ 0.2\\
&\qquad else\ 0).
\end{aligned}$$

So the uncertainty is in the time of the program; it is probably 7.30 pm, but it could be 8.30 pm. Note that an equivalent form of the definition for *choice* is

$$\begin{aligned}
choice = \lambda x.&if\ x = ((21, 7, 2007), (19, 30), ABC)\ then\ 0.8\\
&else\ if\ x = ((21, 7, 2007), (20, 30), ABC)\ then\ 0.2\\
&else\ 0,
\end{aligned}$$

which is the form of the definition that is used below.

Next there is the TV guide that maps occurrences to programs.

$$\begin{aligned}
&tv\_guide : Occurrence \to Program\\
&\forall x.((tv\_guide\ x) =\\
&\qquad if\ x = ((20, 7, 2007), (11, 30), Win)\ then\ (\text{“}NFL\ Football\text{”},\ 60,\ Sport,\ G,\ \text{“}The\ Buffalo\ldots\text{”})\\
&\qquad else\ if\ x = ((21, 7, 2007), (19, 30),\ ABC)\ then\ (\text{“}Seinfeld\text{”}, 30,\ Sitcom,\ PG, \text{“}Kramer\ldots\text{”})\\
&\qquad else\ if\ x = ((21, 7, 2007), (20, 30),\ ABC)\ then\ (\text{“}The\ Bill\text{”}, 50,\ Drama,\ M, \text{“}Sun\ Hill\ldots\text{”})\\
&\qquad else\ if\ x = ((21, 7, 2007), (21, 30),\ ABC)\ then\ (\text{“}Numb3rs\text{”}, 60,\ Crime,\ M, \text{“}When\ldots\text{”})\\
&\qquad else\ if\ x = ((21, 7, 2007), (22, 30),\ ABC)\ then\ (\text{“}ABC\ News\text{”}, 60,\ News,\ G, \text{“}Today\ldots\text{”})\\
&\qquad\qquad \vdots\\
&\qquad else\ (\text{“”}, 0,\ NULL,\ NA, \text{“”})).
\end{aligned}$$

Finally, here is the definition of the function *likes*. (A discussion about how such functions are learned is contained in [8].)

$$\begin{aligned}
&likes : Program \to Density\ \Omega\\
&\forall x.((likes\ x) =\\
&\qquad if\ (projTitle \circ (= \text{“}NFL\ Football\text{”})\ x)\ then\ \lambda y.if\ y = \top\ then\ 1\ else\ 0\\
&\qquad else\ if\ (projGenre \circ (= Sitcom)\ x)\ then\ \lambda y.if\ y = \top\ then\ 0.9\ else\ 0.1\\
&\qquad else\ if\ (projGenre \circ (= Movie)\ x)\ then\ \lambda y.if\ y = \top\ then\ 0.75\ else\ 0.25\\
&\qquad else\ if\ (projGenre \circ (= Documentary)\ x)\ then\ \lambda y.if\ y = \top\ then\ 1\ else\ 0\\
&\qquad else\ if\ (projGenre \circ (= Drama)\ x)\ then\ \lambda y.if\ y = \top\ then\ 0.2\ else\ 0.8\\
&\qquad else\ \lambda y.if\ y = \bot\ then\ 1\ else\ 0).
\end{aligned}$$

Recall that the agent needs to compute the probability that the user likes the program which is on at the occurrence intended by the user. This amounts to computing the value of the term

$$((choice\ \$\ tv\_guide)\ \S\ likes),$$

which is a term of type $\mathit{Density}\ \Omega$. The answer for this computation given by Bach is

$$\lambda z.\mathit{if}\ z = \top\ \mathit{then}\ 0.76\ \mathit{else}\ \mathit{if}\ z = \top\ \mathit{then}\ 0.12\ \mathit{else}\ 0.24,$$

which can be simplified to

$$\lambda z.\mathit{if}\ z = \top\ \mathit{then}\ 0.76\ \mathit{else}\ 0.24.$$

Thus

$$((\mathit{choice}\ \$\ \mathit{tv\_guide})\ \S\ \mathit{likes}) = \lambda z.\mathit{if}\ z = \top\ \mathit{then}\ 0.76\ \mathit{else}\ 0.24$$

is a consequence of the belief base of the agent. On this basis, the agent can now decide whether to recommend the program or not.

The computation is somewhat complicated. To make it easier to understand, we break it up into two steps. Fig. 2 gives the computation of $(\mathit{choice}\ \$\ \mathit{tv\_guide})$ which gives the answer

$$\begin{aligned}&\lambda y.\mathit{if}\ y = (\text{“}\mathit{The\ Bill}\text{”}, 50,\ \mathit{Drama},\ M,\ \text{“}\mathit{Sun\ Hill}\ldots\text{”})\ \mathit{then}\ 0.2\ \mathit{else}\\ &\qquad \mathit{if}\ y = (\text{“}\mathit{Seinfeld}\text{”}, 30,\ \mathit{Sitcom},\ \mathit{PG},\ \text{“}\mathit{Kramer}\ldots\text{”})\ \mathit{then}\ 0.8\ \mathit{else}\ 0,\end{aligned}$$

which is a term of type *Density Program*. Then the second part of the computation in Fig. 3 is that of

$$((\lambda y.\mathit{if}\ y = (\text{“}\mathit{The\ Bill}\text{”}, \ldots)\ \mathit{then}\ 0.2\ \mathit{else}\ \mathit{if}\ y = (\text{“}\mathit{Seinfeld}\text{”}, \ldots)\ \mathit{then}\ 0.8\ \mathit{else}\ 0)\ \S\ \mathit{likes})$$

that has the answer

$$\lambda z.\mathit{if}\ z = \top\ \mathit{then}\ 0.76\ \mathit{else}\ \mathit{if}\ z = \top\ \mathit{then}\ 0.12\ \mathit{else}\ 0.24.$$

The preceding example contradicts a widely held view that standard logics have no direct way of modelling the certainty of assumptions in theories and no direct way of stating the certainty of theorems proved from these (uncertain) assumptions: witness the quote from Williamson [49] in the introduction. For first-order logic, this view seems reasonable. But, for higher-order logic, which surely must be accepted as a ‘classical’ logic, the statement is not true, as the preceding example shows. The higher-orderness of this logic allows one to directly model uncertainty in assumptions and theorems. In the example, reasoning proceeds via computation, which is a form of theorem proving. Further, the chaining of assumptions (both deterministic and probabilistic) in the theory is done by composing functions in different ways. These composition functions are, of course, higher-order functions.

$$\begin{aligned}
&\underline{(\mathit{choice}\ \$\ \mathit{tv\_guide})}\\
&\lambda z.(\mathit{sum2}\ \lambda y.(\underline{(\mathit{tv\_guide}\ y)} = z)\ \mathit{choice})\\
&\lambda z.(\mathit{sum2}\ \lambda y.((\mathit{if}\ y = ((20,7,2007),(11,30),\ \mathit{Win})\ \mathit{then}\ldots) = z)\ \underline{\mathit{choice}})\\
&\lambda z.(\underline{\mathit{sum2}\ \lambda y.((\mathit{if}\ y = ((20,7,2007),(11,30),\ \mathit{Win})\ \mathit{then}\ldots) = z)}\\
&\qquad\qquad\qquad\qquad \underline{\lambda x.\mathit{if}\ x = ((21,7,2007),(19,30),\ \mathit{ABC})\ \mathit{then}\ 0.8\ldots})\\
&\quad\vdots\\
&\lambda z.(\mathit{if}\ z = (\text{“}\mathit{Seinfeld}\text{”},\ldots)\ \mathit{then}\ 0.8\ \mathit{else}\ 0\\
&\qquad +(\underline{\mathit{sum2}\ \lambda y.((\mathit{if}\ y = ((20,7,2007),(11,30),\ \mathit{Win})\ \mathit{then}\ldots) = z)}\\
&\qquad\qquad\qquad\qquad \underline{\lambda x.\mathit{if}\ x = ((21,7,2007),(20,30),\ \mathit{ABC})\ \mathit{then}\ 0.2\ldots})\\
&\quad\vdots\\
&\lambda z.(\mathit{if}\ z = (\text{“}\mathit{Seinfeld}\text{”},\ldots)\ \mathit{then}\ 0.8\ \mathit{else}\ 0\\
&\qquad +\mathit{if}\ z = (\text{“}\mathit{The\ Bill}\text{”},\ldots)\ \mathit{then}\ 0.2\ \mathit{else}\ 0\\
&\qquad +\underline{(\mathit{sum2}\ \lambda y.((\mathit{if}\ y = ((20,7,2007),(11,30),\ \mathit{Win})\ \mathit{then}\ldots) = z)\ \lambda x.0)}\\
&\lambda z.(\mathit{if}\ z = (\text{“}\mathit{Seinfeld}\text{”},\ldots)\ \mathit{then}\ 0.8\ \mathit{else}\ 0\\
&\qquad +\underline{\mathit{if}\ z = (\text{“}\mathit{The\ Bill}\text{”},\ldots)\ \mathit{then}\ 0.2\ \mathit{else}\ 0+0})\\
&\lambda z.(\underline{\mathit{if}\ z = (\text{“}\mathit{Seinfeld}\text{”},\ldots)\ \mathit{then}\ 0.8\ \mathit{else}\ 0+\mathit{if}\ z = (\text{“}\mathit{The\ Bill}\text{”},\ldots)\ \mathit{then}\ 0.2\ \mathit{else}\ 0})\\
&\quad\vdots\\
&\lambda z.\mathit{if}\ z = (\text{“}\mathit{The\ Bill}\text{”},\ldots)\ \mathit{then}\ 0.2\ \mathit{else}\ \mathit{if}\ z = (\text{“}\mathit{Seinfeld}\text{”},\ldots)\ \mathit{then}\ 0.8\ \mathit{else}\ 0
\end{aligned}$$

Fig. 2. Computation of $(\mathit{choice}\ \$\ \mathit{tv\_guide})$.

$$\underline{((\lambda y.\mathit{if}\ y = (\text{“}\mathit{The\ Bill}\text{”}, \ldots)\ \mathit{then}\ 0.2\ \mathit{else}\ \mathit{if}\ y = (\text{“}\mathit{Seinfeld}\text{”}, \ldots)\ \mathit{then}\ 0.8\ \mathit{else}\ 0)\ \S\ \mathit{likes})}$$

$$\begin{aligned}\lambda z.(\mathit{sum1}\ \lambda y.(&\underline{((\lambda y.\mathit{if}\ y = (\text{“}\mathit{The\ Bill}\text{”}, \ldots)\ \mathit{then}\ 0.2\ \mathit{else}}\\ &\underline{\mathit{if}\ y = (\text{“}\mathit{Seinfeld}\text{”}, \ldots)\ \mathit{then}\ 0.8\ \mathit{else}\ 0)\ y)} \times ((\mathit{likes}\ y)\ z)))\end{aligned}$$

$$\begin{aligned}\lambda z.(\mathit{sum1}\ \lambda y.(&\underline{(\mathit{if}\ y = (\text{“}\mathit{The\ Bill}\text{”}, \ldots)\ \mathit{then}\ 0.2\ \mathit{else}}\\ &\underline{\mathit{if}\ y = (\text{“}\mathit{Seinfeld}\text{”}, \ldots)\ \mathit{then}\ 0.8\ \mathit{else}\ 0) \times}((\mathit{likes}\ y)\ z)))\end{aligned}$$

$$\vdots$$

$$\begin{aligned}\lambda z.(\mathit{sum1}\ \lambda y.(&\mathit{if}\ y = (\text{“}\mathit{The\ Bill}\text{”}, \ldots)\ \mathit{then}\ 0.2 \times (\underline{(\mathit{likes}\ (\text{“}\mathit{The\ Bill}\text{”}, \ldots))}\ z)\\ &\mathit{else}\ ((\mathit{if}\ y = (\text{“}\mathit{Seinfeld}\text{”}, \ldots)\ \mathit{then}\ 0.8\ \mathit{else}\ 0) \times ((\mathit{likes}\ y)\ z))))\end{aligned}$$

$$\vdots$$

$$\begin{aligned}\lambda z.(\mathit{sum1}\ \lambda y.(&\mathit{if}\ y = (\text{“}\mathit{The\ Bill}\text{”}, \ldots)\ \mathit{then}\ 0.2 \times \underline{(\lambda y.\mathit{if}\ y = \top\ \mathit{then}\ 0.2\ \mathit{else}\ 0.8\ z)}\\ &\mathit{else}\ ((\mathit{if}\ y = (\text{“}\mathit{Seinfeld}\text{”}, \ldots)\ \mathit{then}\ 0.8\ \mathit{else}\ 0) \times ((\mathit{likes}\ y)\ z))))\end{aligned}$$

$$\begin{aligned}\lambda z.(\mathit{sum1}\ \lambda y.(&\mathit{if}\ y = (\text{“}\mathit{The\ Bill}\text{”}, \ldots)\ \mathit{then}\ \underline{0.2 \times (\mathit{if}\ z = \top\ \mathit{then}\ 0.2\ \mathit{else}\ 0.8)}\\ &\mathit{else}\ ((\mathit{if}\ y = (\text{“}\mathit{Seinfeld}\text{”}, \ldots)\ \mathit{then}\ 0.8\ \mathit{else}\ 0) \times ((\mathit{likes}\ y)\ z))))\end{aligned}$$

$$\vdots$$

$$\begin{aligned}\lambda z.(\mathit{sum1}\ \lambda y.(&\mathit{if}\ y = (\text{“}\mathit{The\ Bill}\text{”}, \ldots)\ \mathit{then}\ (\mathit{if}\ z = \top\ \mathit{then}\ 0.04\ \mathit{else}\ 0.16)\ \mathit{else}\\ &\underline{((\mathit{if}\ y = (\text{“}\mathit{Seinfeld}\text{”}, \ldots)\ \mathit{then}\ 0.8\ \mathit{else}\ 0) \times ((\mathit{likes}\ y)\ z))}))\end{aligned}$$

$$\vdots$$

$$\begin{aligned}\lambda z.\underline{(\mathit{sum1}\ \lambda y.(}&\underline{\mathit{if}\ y = (\text{“}\mathit{The\ Bill}\text{”}, \ldots)\ \mathit{then}\ (\mathit{if}\ z = \top\ \mathit{then}\ 0.04\ \mathit{else}\ 0.16)\ \mathit{else}}\\ &\underline{\mathit{if}\ y = (\text{“}\mathit{Seinfeld}\text{”}, \ldots)\ \mathit{then}\ (\mathit{if}\ z = \top\ \mathit{then}\ 0.72\ \mathit{else}\ 0.08)\ \mathit{else}\ 0))}\end{aligned}$$

$$\vdots$$

$$\lambda z.(\underline{(\mathit{if}\ z = \top\ \mathit{then}\ 0.04\ \mathit{else}\ 0.16) + (\mathit{if}\ z = \top\ \mathit{then}\ 0.72\ \mathit{else}\ 0.08)})$$

$$\lambda z.\mathit{if}\ z = \top\ \mathit{then}\ \underline{(0.04 + 0.72)}\ \mathit{else}\ (\mathit{if}\ z = \top\ \mathit{then}\ 0.04\ \mathit{else}\ 0.16 + 0.08)$$

$$\lambda z.\mathit{if}\ z = \top\ \mathit{then}\ 0.76\ \mathit{else}\ (\underline{\mathit{if}\ z = \top\ \mathit{then}\ 0.04\ \mathit{else}\ 0.16 +}\ 0.08)$$

$$\lambda z.\mathit{if}\ z = \top\ \mathit{then}\ 0.76\ \mathit{else}\ \underline{(\mathit{if}\ z = \top\ \mathit{then}\ (+\ 0.04)\ \mathit{else}\ (+\ 0.16)\ 0.08)}$$

$$\lambda z.\mathit{if}\ z = \top\ \mathit{then}\ 0.76\ \mathit{else}\ \mathit{if}\ z = \top\ \mathit{then}\ \underline{(0.04 + 0.08)}\ \mathit{else}\ (0.16 + 0.08)$$

$$\lambda z.\mathit{if}\ z = \top\ \mathit{then}\ 0.76\ \mathit{else}\ \mathit{if}\ z = \top\ \mathit{then}\ 0.12\ \mathit{else}\ \underline{(0.16 + 0.08)}$$

$$\lambda z.\mathit{if}\ z = \top\ \mathit{then}\ 0.76\ \mathit{else}\ \mathit{if}\ z = \top\ \mathit{then}\ 0.12\ \mathit{else}\ 0.24$$

Fig. 3. Computation of $((\lambda y.\mathit{if}\, y = (\text{“}\mathit{The\ Bill}\text{”}, \ldots)\ \mathit{then}\ 0.2\ \mathit{else}\ \ldots)\ \S\ \mathit{likes})$. The last few steps of the computation make crucial use of Eqs. (4) and (5) in the standard equality theory.

5. More probabilistic reasoning examples

We show in this section how the different kinds of probabilistic reasoning problems being tackled in the first-order probabilistic languages literature can be handled in our framework. The aim is to demonstrate the generality of our approach; most existing first-order systems can only deal with a subset of these problems.

The example shown in Section 4 illustrates the kind of problems being investigated in formalisms like stochastic logic programs [36]. Section 5.1 presents a neat solution to a classical puzzle in probability theory. Section 5.2 gives a solution to a generative probabilistic model problem described in [34]. Section 5.3 uses an example from [15] and [23] to show how logic can be used to obtain compact representations of large Bayesian networks. Section 5.4 shows an example of inference with continuous densities. The general strategy employed in all these problems is to write down a joint distribution (corresponding to a probabilistic network) and then reason with it.

5.1. Monty Hall problem

The problem can be stated as follows. An honest game-show host has placed a car behind one of three doors. There is a goat behind each of the other doors. You have no prior knowledge that allows you to distinguish among the doors. “First you pick a door”, he says. “Then I will open one of the other doors to reveal a goat. After I’ve shown you the

$$\begin{tikzcd} \boxed{carDoor} \arrow[r] & \boxed{hostDoor} & \boxed{initChoice} \arrow[l] \end{tikzcd}$$

Fig. 4. A Bayesian network for the Monty Hall problem.

$$\begin{tikzcd} \boxed{numOfBalls} \arrow[r] & \boxed{setOfBalls} \arrow[r] & \boxed{ballsDrawn_d} \arrow[r] & \boxed{observations} \end{tikzcd}$$

Fig. 5. A Bayesian network for the balls problem.

goat, you make a final decision whether to stick to your initial choice or to switch to the remaining door. You win whatever is behind the door."

Fig. 4 shows a Bayesian network that models the above. The random variable *carDoor* gives the door with the car behind; *initChoice* gives the door initially chosen by the player; *hostDoor* gives the door opened by the host. Clearly, the host's choice of doors is dependent on the actual values of *carDoor* and *initChoice*, and that *carDoor* and *initChoice* are independent of each other.

We have three constants $D_1, D_2, D_3 : Door$ representing the three doors. The following specifies the Bayesian network in full.

$$\begin{aligned}
&carDoor : Density\, Door\\
&(carDoor\ x) = 1/3\\
&initChoice : Density\, Door\\
&(initChoice\ x) = 1/3\\
&hostDoor : Door \to Door \to Density\, Door\\
&(hostDoor\ x_c\ x_i) = \mathit{if}\ (x_c = x_i)\ \mathit{then}\ \lambda x.\mathit{if}\ (x \neq x_i)\ \mathit{then}\ 0.5\ \mathit{else}\ 0\\
&\qquad\qquad\qquad\qquad \mathit{else}\ \lambda x.(\mathbb{I}\ ((x \neq x_c) \wedge (x \neq x_i)))\\
&joint : Density\ (Door \times Door \times Door)\\
&(joint\ (x_c, x_i, x_h)) = (carDoor\ x_c) \times (initChoice\ x_i) \times (hostDoor\ x_c\ x_i\ x_h)
\end{aligned}$$

Let D_i be the door chosen by the player, D_h the door revealed by the host, and D_r the remaining unopened door. We only need to know the posterior probability of the car door given D_i and D_h to know how to act and this is given by the density

$$\begin{aligned}
&f_{cd} : Density\ Door\\
&f_{cd} = \lambda x.\frac{(joint\ (x, D_i, D_h))}{(joint\ (D_1, D_i, D_h)) + (joint\ (D_2, D_i, D_h)) + (joint\ (D_3, D_i, D_h))}.
\end{aligned}$$

Given the above, Bach returns the following results

$$(f_{cd}\ D_i) = 1/3, \quad (f_{cd}\ D_r) = 2/3, \quad \text{and} \quad (f_{cd}\ D_h) = 0,$$

showing that the player should indeed switch to double her chance of winning.

5.2. Urn and balls

We next look at an interesting problem from [34]. We want to model the following scenario. An urn contains an unknown number of balls – say, a number chosen from a Poisson distribution with mean 6. Balls are assigned the colour blue and green with equal probability. We draw some balls from the urn, observing the colour of each and replacing it. We cannot tell two identically coloured balls apart. Furthermore, observed colours are wrong with probability 0.2.

It was claimed in [34] that this problem is hard to model in most existing first-order probabilistic languages because there is an unknown and unbounded number of balls in the urn. Interestingly, there is a simple Bayesian network for the problem (Fig. 5) if we can define densities over structured objects like sets and lists.

The Bayesian network can be understood as follows. We first pick a number n from the appropriate Poisson distribution. A set s of balls of size n is then constructed. We represent a ball by an integer identifier and its colour:

$Ball = Int \times Colour$. The balls in s are labelled 1 to n, and the colours are picked randomly. Given s, we then construct a list consisting of d balls by drawing successively at random with replacement from s. The observed colours of the drawn balls are recorded. The Bayesian network is specified in full as follows.

$$
\begin{aligned}
& colour : Colour \to \Omega \\
& (colour\ x) = (x = Blue) \vee (x = Green) \\
& numOfBalls : Density\ Int \\
& (numOfBalls\ x) = (poisson\ 6\ x) \\
& poisson : Int \to Density\ Int \\
& (poisson\ x\ y) = e^{-x} x^{y}/y! \\
& setOfBalls : Int \to Density\ \{Ball\} \\
& (setOfBalls\ n\ s) = if\ \exists x_1 \cdots \exists x_n.((colour\ x_1) \wedge \cdots \wedge (colour\ x_n) \wedge (s = \{(1, x_1), \ldots, (n, x_n)\})\ then\ 0.5^n\ else\ 0 \\
& ballsDrawn : Int \to \{Ball\} \to Density\ (List\ Ball) \\
& (ballsDrawn\ d\ s\ x) = if\ \exists x_1 \cdots \exists x_d.((s\ x_1) \wedge \cdots \wedge (s\ x_d) \wedge (x = [x_1, \ldots, x_d]))\ then\ (card\ s)^{-d}\ else\ 0 \\
& observations : (List\ Ball) \to Density\ (List\ Colour) \\
& (observations\ x\ y) = if\ (length\ x) = (length\ y)\ then\ (obsProb\ x\ y)\ else\ 0 \\
& obsProb : (List\ Ball) \to (List\ Colour) \to Real \\
& (obsProb\ []\ []) = 1 \\
& (obsProb\ (\#\ (x_1, y_1)\ z_1)\ (\#\ y_2\ z_2)) = (if\ (y_1 = y_2)\ then\ 0.8\ else\ 0.2) \times (obsProb\ z_1\ z_2) \\
& joint : Int \to Density\ (Int \times \{Ball\} \times (List\ Ball) \times (List\ Colour)) \\
& (joint\ d\ (n, s, x, y)) = (numOfBalls\ n) \times (setOfBalls\ n\ s) \times (ballsDrawn\ d\ s\ x) \times (observations\ x\ y)
\end{aligned}
$$

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 easily given.

We can compute marginalisations of the given density to obtain answers to different questions. For example, the following gives the probability that the number of balls in the urn is m after we have seen the colours $[o_1, o_2, \ldots, o_d]$ from d draws:

$$\frac{1}{K} \sum_{s} \sum_{l} (joint\ d\ (m, s, l, [o_1, o_2, \ldots, o_d])), \tag{9}$$

where K is a normalisation constant, s ranges over the set $\{s \mid (setOfBalls\ m\ s) > 0\}$, and l ranges over the set $\{l \mid (ballsDrawn\ d\ s\ l) > 0\}$. The expression (9) is an informal mathematical expression. The corresponding term in the logic is

$$
\begin{aligned}
& \frac{1}{K} \times (sum3\ \lambda s.((setOfBalls\ m\ s) > 0) \\
& \qquad \lambda s.(sum3\ \lambda l.((ballsDrawn\ d\ s\ l) > 0)\ \lambda l.(joint\ d\ (m, s, l, [o_1, o_2, \ldots, o_d])))).
\end{aligned}
$$

The elements of the two sets

$$\lambda s.((setOfBalls\ m\ s) > 0) \text{ and } \lambda l.((ballsDrawn\ d\ s\ l) > 0)$$

are automatically enumerated by Bach during computations via rewrites involving Eq. (7) and a few others in the standard equality theory. It is the extensional forms of these two sets that are needed by the definition of *sum3* and we get that transformation for free with the way Bach works.

Fig. 6 shows the posterior distribution of m after drawing ten and fifteen balls and observing that they are all blue. Consistent with intuition, the lower numbers of m become increasingly probable as more blue balls are observed.

Other questions like "What is the probability that the same ball was drawn twice given observations $[o_1, o_2, \ldots, o_d]$?" can be answered in a similar way.

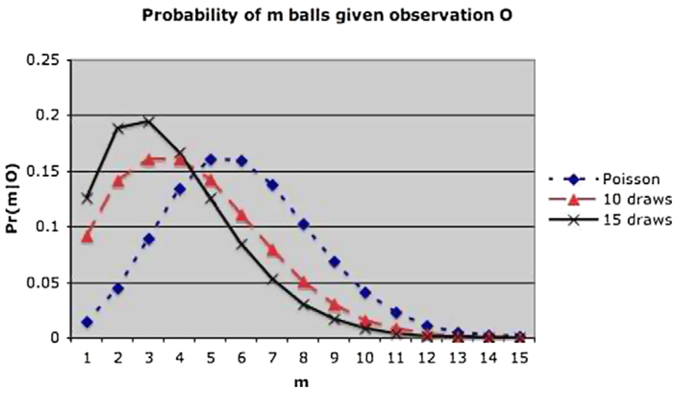

Fig. 6. The posterior distributions of the number of balls in the urn after some observations. The Poisson curve is the prior distribution on the number of balls.

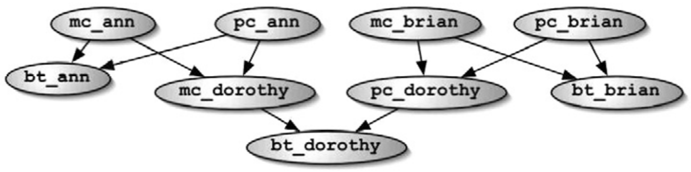

Fig. 7. A Bayesian network modelling the inheritance of blood types within a family. The variables mc_X and pc_X represent respectively the m and p-chromosomes of person X. The variable bt_X represents the blood type of person X.

5.3. Genetics

We next look at an application taken from [23] (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 the mother, and one, the p-chromosome, inherited from the father. Fig. 7 taken from [23] shows a Bayesian network modelling the inheritance of blood types within a particular family.

Note that although different families are associated with different Bayesian networks, each instance of these networks has 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 general strategy employed by formalisms like BLP [23].

We now show one way the Bayesian logic program given in [23, Fig. 1.4] for the Bayesian network shown in Fig. 7 can be encoded in a theory. We define a probability density function 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 the blood type of each person in the family. 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.

$$A, B, O : Chromosome$$

$$A, B, AB, O : BloodType$$

$$Ann,\ Brian,\ Dorothy,\ Unknown : Person$$

$$family = [Ann,\ Brian,\ Dorothy]$$

$$(mother\ x) = if\ (x = Dorothy)\ then\ Ann\ else\ Unknown$$

$$(father\ x) = if\ (x = Dorothy)\ then\ Brian\ else\ Unknown$$

$$
\begin{aligned}
&joint : Density\ (List\ Chromosome) \times (List\ Chromosome) \times (List\ BloodType)\\
&(joint\ (x_1, x_2, x_3)) = (joint_2\ \ family\ (x_1, x_2, x_3)\ (x_1, x_2))\\
&joint_2 : (List\ Person) \to (List\ Chromosome) \times (List\ Chromosome)\ \times\ (List\ BloodType)\\
&\qquad\qquad \to (List\ Chromosome) \times (List\ Chromosome) \to Real\\
&(joint_2\ []\ ([],\ [],\ [])\ l) = 1\\
&(joint_2\ (\#\ p\ t_0)\ ((\#\ x_1\ t_1), (\#\ x_2\ t_2), (\#\ x_3\ t_3))\ l) =\\
&\qquad\qquad\qquad (mc\ p\ l\ x_1) \times (pc\ p\ l\ x_2) \times (bt\ x_1\ x_2\ x_3) \times (joint_2\ t_0\ (t_1, t_2, t_3)\ l)\\
&mc : Person \to (List\ Chromosome) \times (List\ Chromosome) \to Chromosome \to Real\\
&(mc\ p\ (l_1, l_2)\ x) = if\ (mother\ p) = Unknown\ then\ 1/3\\
&\qquad\qquad\qquad\qquad else\ (mc_2\ (getVal\ (mother\ p)\ l_1)\ (getVal\ (mother\ p)\ l_2)\ x)\\
&mc_2 : Chromosome \to Chromosome \to (Density\ Chromosome)\\
&(mc_2\ A\ A\ z) = if\ (z = A)\ then\ 0.98\ else\ 0.01\\
&(mc_2\ B\ A\ z) = if\ (z = B)\ then\ 0.98\ else\ 0.01 \quad \ldots\ \text{etc.}\\
&pc : Person \to (List\ Chromosome) \times (List\ Chromosome) \to Chromosome \to Real\\
&(pc\ p\ (l_1, l_2)\ x) = if\ (father\ p) = Unknown\ then\ 1/3\\
&\qquad\qquad\qquad\qquad else\ (pc_2\ (getVal\ (father\ p)\ l_1)\ (getVal\ (father\ p)\ l_2)\ x)\\
&pc_2 : Chromosome \to Chromosome \to (Density\ Chromosome)\\
&(pc_2\ A\ A\ z) = if\ (z = A)\ then\ 0.98\ else\ 0.01\\
&(pc_2\ B\ A\ z) = if\ (z = A)\ then\ 0.98\ else\ 0.01 \quad \ldots\ \text{etc.}\\
&bt : Chromosome \to Chromosome \to (Density\ BloodType)\\
&(bt\ A\ A\ z) = if\ (z = A)\ then\ 0.97\ else\ 0.01\\
&(bt\ B\ A\ z) = if\ (z = AB)\ then\ 0.97\ else\ 0.01 \quad \ldots\ \text{etc.}
\end{aligned}
$$

The function *getVal* returns the nth element in a list indexed by a person. The functions *bt*, mc_2 and pc_2 are reproductions of the conditional probability tables given in [23]. It is assumed that all the arguments to the function *joint* have the same length.

Just like in [23] and [15], the m-chromosome, p-chromosome, and blood type of each person is a random variable in the given joint distribution. One can verify that the term $(joint\ ([x_1, x_2, x_3], [y_1, y_2, y_3], [z_1, z_2, z_3]))$, when expanded, expresses the exact same factorisation of the distribution on the random variables as that given by the Bayesian network in Fig. 7.

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 the mathematical expression

$$\frac{\sum_{x_i, y_i, z_i} (joint\ ([x_1, A, x_2], [O, y_1, y_2], [z_1, z_2, A]))}{\sum_{x_i, y_i, z_i} (joint\ ([x_1, A, x_2], [O, y_1, y_2], [z_1, z_2, z_3]))}, \tag{10}$$

where $x_i \in \{A, B, O\}$, $y_i \in \{A, B, O\}$, and $z_i \in \{A, B, AB, O\}$. It is straightforward to write down the term representation of an expression like (10) and compute its value.

5.4. Wages

We next present an example taken from [37] to illustrate inference with continuous densities. Suppose Mr Dobson has a job that pays \$10 an hour and he is expected to work 40 hours per week. However, he is not guaranteed 40 hours every week. He estimates the number of hours worked in a week to be normally distributed with mean 40 and standard deviation 5. He has not fully investigated the benefits such as bonus pay and nontaxable deductions such as contributions to a retirement program. He estimates these other influences on his gross taxable weekly income also

$$\begin{tikzcd}\boxed{\mathit{weeklyHours}} \arrow[r] & \boxed{\mathit{weeklyIncome}}\end{tikzcd}$$

Fig. 8. A Bayesian network for Mr Dobson's weekly income.

to be normally distributed with mean 0 (i.e. he believes they approximately offset each other) and standard deviation 30. Furthermore, Mr Dobson assumes that these other benefits influences are independent of the number of hours he works each week. Mr Dobson's gross taxable weekly income is simply the sum of his paid work and the extra benefits. The Bayesian network shown in Fig. 8 can be used to capture the scenario just described.

One can show (see [37]) that the conditional density of Mr Dobson's gross taxable weekly income given the number x of hours worked is normally distributed with mean $10x$ and standard deviation 30. This leads us to the following theory:

$$\begin{aligned}
&\mathit{gaussian} : \mathit{Real} \to \mathit{Real} \to \mathit{Density\ Real}\\
&(\mathit{gaussian}\ \mu\ \sigma) = \lambda x.\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}\\
&\mathit{weeklyHours} : \mathit{Density\ Real}\\
&(\mathit{weeklyHours}\ x) = (\mathit{gaussian}\ 40\ 5\ x)\\
&\mathit{weeklyIncome} : \mathit{Real} \to \mathit{Density\ Real}\\
&(\mathit{weeklyIncome}\ x\ \ y) = (\mathit{gaussian}\ y\ 30\ 10x)\\
&\mathit{joint} : \mathit{Density}\ (\mathit{Real} \times \mathit{Real})\\
&(\mathit{joint}\ (x, y)) = (\mathit{weeklyHours}\ x) \times (\mathit{weeklyIncome}\ x\ y),
\end{aligned}$$

where we have exploited the fact that

$$(\mathit{gaussian}\ \mu\ \sigma\ x) = (\mathit{gaussian}\ x\ \sigma\ \mu)$$

in the definition of *weeklyIncome*.

Given the above, one can try to find out, for example, the (unconditional) density of Mr Dobson's weekly income, which is given by the term

$$\lambda y.(\mathit{integrate}\ \lambda x.(\mathit{joint}\,(x, y))), \tag{11}$$

where *integrate* : $(a \to \mathit{Real}) \to \mathit{Real}$ is the integration function. By introducing into the theory additional known facts about normal distributions like

$$\begin{aligned}
&(\mathit{gaussian}\ \mu\ \sigma\ ax) = \frac{1}{a} \times \left(\mathit{gaussian}\ \frac{\mu}{a}\ \frac{\sigma}{a}\ x\right) \text{ and}\\
&(\mathit{integrate}\ \lambda x.((\mathit{gaussian}\ \mu_1\ \sigma_1\ x) \times (\mathit{gaussian}\ y\ \sigma_2\ x))) = \left(\mathit{gaussian}\ \mu_1\ \sqrt{\sigma_1^2 + \sigma_2^2}\ y\right),
\end{aligned}$$

Bach can simplify (11) to $\lambda y.(\mathit{gaussian}\ 400\ \sqrt{3400}\ y)$, showing that Mr Dobson's taxable weekly income is normally distributed with mean 400 and standard deviation 58.3.

6. Discussion

General approach

We begin our discussion by comparing what is proposed in this paper with existing (first-order) 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. 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 earlier 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. 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.

On the expressiveness of higher-order logic

Higher-order logic (also known as type theory) is a language that was developed for formalising mathematics [3,26]. It is thus not surprising that everyday mathematical concepts including probabilistic ones can be represented in higher-order logic. We illustrate this point in the following.

We will first show how Bayesian networks and Markov random fields can be represented in higher-order logic. It is well known [6,22] that the joint distribution given by a Bayesian network with K nodes is

$$p(\boldsymbol{x}) = \prod_{k=1}^{K} p(x_k \mid pa_k), \tag{12}$$

where $\boldsymbol{x} \equiv \{x_1, \ldots, x_K\}$ is the set of random variables associated with the nodes of the graph, and pa_k denotes the set of parents of x_k. Each factor $p(x_k \mid pa_k)$ in (12) is represented by either a conditional probability table or more generally a function that takes n arguments, where n is the cardinality of pa_k, and returns a density over the domain of x_k. The joint distribution associated with a Markov random field is given by an expression of the form

$$p(\boldsymbol{x}) = \frac{1}{Z} \prod_{C} \psi_C(\boldsymbol{x}_C), \tag{13}$$

where $\boldsymbol{x}$ is the set of random variables associated with the nodes of the graph, Z is a normalisation constant, each $\boldsymbol{x}_C \subseteq \boldsymbol{x}$ is a maximal clique of the graph, and ψ_C is a potential function over $\boldsymbol{x}_C$. Clearly, expressions like (12) and (13) can be written down directly in higher-order logic. Some specific examples of this translation from graphical models into theories are shown earlier in Section 5.

We next examine the syntactic features of a few representative examples of (first-order) probabilistic logics and show how they can be expressed in higher-order logic. In the logic of [16], one can write expressions such as $w(\varphi) < 1/3$ and $w(\varphi) \geqslant 2w(\psi)$, where φ and ψ are (first-order) formulas. The first statement means that "φ has probability less than $1/3$" and the second means "φ is at least twice as probable as ψ". To write such expressions in higher-order logic, one replaces the formula φ in the logic of [16] by a density on the booleans denoted by φ'. Then $w(\varphi) < 1/3$ is written in higher-order logic as $(\varphi'\ \top) < 1/3$ and $w(\varphi) \geqslant 2w(\psi)$ is written as $(\varphi'\ \top) \geqslant 2(\psi'\ \top)$.

The actual meaning attached to the word probability is often important in studies on probabilistic logics. We note here that both the frequentist and degree-of-belief interpretations of probability are realised in the same logical form in our setting, that of a density. This point is now illustrated with some examples introduced in [20]. Halpern described several approaches to giving semantics to first-order logics of probability. His type 1 probability structure gives a logic with a frequentist semantics. A typical statement that can be made in that setting is “The probability that a randomly chosen bird flies is greater than 0.9”. Halpern’s type 2 probability structure gives a logic with a degree-of-belief semantics. A typical statement that can be made in this second setting (but not the first) is “The probability that Tweety (a particular bird) flies is greater than 0.9”. Finally, his type 3 probability structure gives a logic that accommodates both the frequentist and degree-of-belief interpretations of probability. A typical statement that can be made in this last setting is “The probability that Tweety flies is greater than the probability that a randomly chosen bird flies”. All of the statements above can be formulated in higher-order logic as follows:

$$aRandomBird : Density\ Bird$$

$$flies : Bird \rightarrow Density\ \Omega$$

$$aRandomBird = \lambda x.(1/m) \quad \text{-- where } m \text{ is the total number of birds in the domain} \tag{14}$$

$$((aRandomBird\ \S\ flies)\ \top) > 0.9 \tag{15}$$

$$((flies\ Tweety)\ \top) > 0.9 \tag{16}$$

$$((flies\ Tweety)\ \top) > ((aRandomBird\ \S\ flies)\ \top), \tag{17}$$

where (14)–(15) captures the first frequentist statement above, (16) captures the second degree-of-belief statement, and (17) captures the last mixed-semantics statement.

We will now show how probabilistic logic programs [33,40] can be represented in our setting, using an example motivated by [13]. We want to capture statements in probabilistic logic programming like the following:

$$priceDrops(C) : [0.4, 0.9] \longleftarrow (ceoSellsStock(C) \vee ceoRetires(C)) : [0.6, 1], \tag{18}$$

which says that the stock of a company will drop with probability between 0.4 and 0.9 if the probability that the CEO of the company will sell the stock or that he will retire is more than 0.6. Note that the antecedent of (18) talks about the probability of a joint event, the exact details of which is not specified. We could duplicate the different forms of disjunction and conjunction introduced in [13] to capture different assumptions on the relationship between *ceoSellsStock* and *ceoRetires*. For our purpose here, we will assume that whether or not a CEO retires is an independent company-specific event, but that a retiring CEO is more likely than not to sell his stocks. The following theory captures the scenario just described. Formulas (19)–(22) capture the additional assumption we have made. Formula (23) is a direct translation of (18) above.

$$ceoRetires : Stock \rightarrow Density\ \Omega$$

$$(ceoRetires\ WOW) = \lambda x.if\ (x = \top)\ then\ 0.9\ else\ 0.1 \tag{19}$$

$$(ceoRetires\ PBL) = \lambda x.if\ (x = \top)\ then\ 0.05\ else\ 0.95 \tag{20}$$

$$\ldots \text{ etc.}$$

$$ceoSellsStock : \Omega \rightarrow Density\ \Omega$$

$$\begin{aligned}(ceoSellsStock\ y) = if\ (y = \top)\ then\ &\lambda x.if\ (x = \top)\ then\ 0.8\ else\ 0.2\\ else\ &\lambda x.if\ (x = \top)\ then\ 0.1\ else\ 0.9\end{aligned} \tag{21}$$

$$ceoJoint : Stock \rightarrow Density\ \Omega \times \Omega$$

$$(ceoJoint\ s\ (x, y)) = (ceoRetires\ s\ x) \times (ceoSellsStock\ x\ y) \tag{22}$$

$$priceDrops : Stock \rightarrow Density\ \Omega$$

$$\begin{aligned}&((ceoJoint\ s\ (\top, \top)) + (ceoJoint\ s\ (\top, \bot)) + (ceoJoint\ s\ (\bot, \top)) > 0.6)\\ &\quad \longrightarrow (0.4 < (priceDrops\ s\ \top) < 0.9)\end{aligned} \tag{23}$$

The above demonstrates the expressive power of higher-order logic. However, the interesting question is not so much what can be represented in higher-order logic, but what kind of reasoning can be effectively supported in higher-order logic. Being able to represent the desired probabilistic concepts directly is clearly a good start. We have seen in this paper how equational reasoning can be used to support a wide range of probabilistic reasoning tasks. The paragraph on future work below outlines what currently is not supported in our system and what needs to be done to overcome existing limitations.

Complexity issues

We now discuss the complexity of probabilistic reasoning in Bach. Subject to proper encodings of problems, the complexity of specific probabilistic-reasoning tasks in Bach can be analysed independently of Bach. This is because Bach is effectively a programming language and, as with Prolog or Haskell, carrying out a computational procedure of a certain complexity in Bach does not result in a computation that is more complex than the original procedure, nothing beyond a small constant factor anyway. This means many known complexity results continue to hold in our setting. In particular, hardness results for general exact inference in graphical models [11,25] still hold, unsurprisingly. More pleasantly, the many positive results for inference on densities with 'nice' factorisations [2,24] also continue to hold in our setting.

We remark also that we are not trying to give a complete axiomatisation of probabilistic reasoning in higher-order logic in this paper; the results of [20] and others demonstrate that this cannot be done in general. What we are proposing instead is that in many practical applications of interest, one can write down a theory in higher-order logic that captures neatly both the probabilistic and non-probabilistic elements of the underlying application domain. Further, reasoning efficiently with such a theory is often easy. Indeed, we believe a great deal of interesting probabilistic reasoning can be carried out this way.

Other related work

In [17], the authors explore ways of defining probability distributions over basic terms, a class of terms in higher-order logic introduced in [29] for the specific purpose of representing individuals in applications. The class of distributions identified in [17] are all defined by induction on the structure of basic terms. An efficient sampling algorithm is also given for these distributions. We remark that the use of higher-order logic in [17] is restricted to the definition of basic terms and is quite different to our use of higher-order logic for representing probabilistic concepts. In particular, the distributions on basic terms identified in [17] are not actually defined and manipulated in higher-order logic, although the techniques of this paper show that this is indeed possible. We note also that the class of distributions identified in [17] forms only a small subset of the kind of distributions that can be represented in higher-order logic.

There are also related works on adding probabilistic reasoning support to functional programming languages. These come in the form of language extensions to the λ-calculus. In [44], the authors show how probability distributions can be captured using monads. They also introduced a simple language called measure terms that allows certain operations on (discrete) distributions to be computed efficiently. The measure terms is a subset of the language we use for representing and manipulating densities in this paper. In [42], a probabilistic language based on sampling functions is presented. The language supports all kinds of probability distributions and exploits the fact that sampling functions form a so-called state monad.

In [1], the Haskell language is used to describe a family of statistical models based on the MML principle. The generalisation of the various models, and the natural mappings between them, are shown by the use of Haskell classes, types, and functions.

Current limitations and future work

We end this section by stating the limitations of what has been done in this paper and suggest some future work.

All the examples given in this paper are solved via exact inference. In some problems, exact inference can be extremely expensive. For example, it would take a long time to compute in the context of Section 5.2 the exact answer to the question "What is the probability that the urn contains 30 balls given that 100 balls are drawn and 75 appear blue and 25 appear green?". Efficient approximate algorithms need to be developed to handle these questions. The main technical problem is coming up with efficient ways of sampling from different classes of terms.

The type of reasoning problems that can be solved using our present system is limited also by the fact that the version of Bach we use here only performs equational reasoning. So, for example, inequality type of reasoning cannot be easily solved at present. The computation system described in this paper can be extended with constraints-processing capabilities to help with such problems. Also, the full Bach system as defined in [30] contains a tableaux theorem prover as a subsystem. This full system, once completed, can be used to tackle a wider class of probabilistic reasoning problems.

We have not discussed the problem of learning in this paper. We are currently pursuing ways of upgrading the Alkemy system [29,31,38] to learn probabilistic theories, building on the work of [7] and maximum entropy methods.

7. Conclusion

The foundations of probability theory can be stated in the language of mathematics. It should thus come as no surprise that probability and probabilistic reasoning can be incorporated naturally and directly in higher-order logic, which is essentially the formal version of the (informal) everyday language of mathematics. We believe it is better to start off from higher-order logic instead of first-order logic in the search for a language that tightly integrates probability and logic. Indeed, taking this route, one may discover that there is actually no need to invent any new machinery to achieve the desired integration.

Acknowledgements

We thank Marcus Hutter, Kristian Kersting, Brian Milch, Tim Sears, Will Uther, Joel Veness, and Yang Wang for helpful discussions. NICTA is funded through the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

References

[1] L. Allison, Models for machine learning and data mining in functional programming, Journal of Functional Programming 15 (1) (2005) 15–32.
[2] S.M. Aji, R.J. McEliece, The generalized distributive law, IEEE Transactions on Information Theory 46 (2) (2000) 325–343.
[3] P.B. Andrews, An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof, Academic Press, 1986.
[4] C. Baral, M. Gelfond, J. Nelson Rushton, Probabilistic reasoning with answer sets, in: Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning, 2004, pp. 21–33.
[5] P. Billingsley, Probability and Measure, second ed., Wiley, 1986.
[6] C.M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.
[7] W.L. Buntine, A theory of learning classification rules, PhD thesis, School of Computing Science, University of Technology, Sydney, 1992.
[8] J.J. Cole, M. Gray, J.W. Lloyd, K.S. Ng, Personalisation for user agents, in: F. Dignum et al., (Ed.), Proceedings of the 4th International Joint Conference on Autonomous Agents and Multi Agent Systems, 2005, pp. 603–610.
[9] A. Church, A formulation of the simple theory of types, Journal of Symbolic Logic 5 (1940) 56–68.
[10] K. Clark, Negation as failure, in: H. Gallaire, J. Minker (Eds.), Logic and Databases, Plenum Press, 1978, pp. 293–322.
[11] G.F. Cooper, The computational complexity of probabilistic inference using Bayesian belief networks, Artificial Intelligence 42 (2–3) (1990) 393–405.
[12] L. De Raedt, K. Kersting, Probabilistic logic learning, SIGKDD Explorations 5 (1) (2003) 31–48.
[13] A. Dekhtyar, V.S. Subrahmanian, Hybrid probabilistic programs, Journal of Logic Programming 43 (3) (2000) 187–250.
[14] R.M. Dudley, Real Analysis and Probability, Cambridge University Press, 2002.
[15] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer, Learning probabilistic relational models, in: Proceedings of the 16th International Joint Conference on Artificial Intelligence, 1999, pp. 1300–1307.
[16] R. Fagin, J.Y. Halpern, Reasoning about knowledge and probability, Journal of the ACM 41 (2) (1994) 340–367.
[17] E. Gyftodimos, P.A. Flach, Combining Bayesian networks with higher-order data representations, in: 6th International Symposium on Intelligent Data Analysis, 2005 pp. 145–156.
[18] M. Gelfond, V. Lifschitz, Classical negation in logic programs and disjunctive databases, New Generation Computing 9 (1991) 365–387.
[19] A. Hájek, Probability, logic and probability logic, in: L. Goble (Ed.), The Blackwell Guide to Philosophical Logic, Blackwell, 2001, pp. 362–384, Chapter 16.
[20] J.Y. Halpern, An analysis of first-order logics of probability, Artificial Intelligence 46 (3) (1990) 311–350.
[21] J.Y. Halpern, Reasoning about Uncertainty, MIT Press, 2003.
[22] M.I. Jordan, Graphical models, Statistical Science 19 (2004) 140–155.

[23] K. Kersting, L. De Raedt, Bayesian logic programming: Theory and tool, in: L. Getoor, B. Taskar (Eds.), Introduction to Statistical Relational Learning, MIT Press, 2007, Chapter 10.

[24] F.R. Kschischang, B.J. Frey, H.-A. Loeliger, Factor graphs and the sum-product algorithm, IEEE Transactions on Information Theory 47 (2) (2001).

[25] V. Kolmogorov, R. Zabih, What energy functions can be minimized via graph cuts? IEEE Transactions on Pattern Analysis and Machine Intelligence 26 (2) (2004) 147–159.

[26] D. Leivant, Higher-order logic, in: D.M. Gabbay, C.J. Hogger, J.A. Robinson, J. Siekmann (Eds.), Handbook of Logic in Artificial Intelligence and Logic Programming, vol. 2, Oxford University Press, 1994, pp. 230–321.

[27] J.W. Lloyd, Programming in an integrated functional and logic language, Journal of Functional and Logic Programming 3 (1999).

[28] J.W. Lloyd, Knowledge representation, computation, and learning in higher-order logic, Available at http://rsise.anu.edu.au/~jwl/, 2002.

[29] J.W. Lloyd, Logic for Learning: Learning Comprehensible Theories from Structured Data, Springer, 2003.

[30] J.W. Lloyd, Knowledge representation and reasoning in modal higher-order logic, Available at http://rsise.anu.edu.au/~jwl, 2007.

[31] J.W. Lloyd, K.S. Ng, Learning modal theories, in: S. Muggleton, R. Otero, A. Tamaddoni-Nezhad (Eds.), Proceedings of the 16th International Conference on Inductive Logic Programming, in: LNAI, vol. 4455, 2007, pp. 320–334.

[32] J.W. Lloyd, K.S. Ng, Probabilistic and logical beliefs, in: M. Dastani, et al. (Eds.), Proceedings of the Workshop on Languages, Methodologies, and Development Tools for Multi-Agent Systems (LADS), 2007, pp. 1–16.

[33] L.V.S. Lakshmanan, F. Sadri, Modeling uncertainty in deductive databases, in: D. Karagianni (Ed.), Proceedings of the International Conference on Database and Expert Systems Applications, DEXA'94, 1994, pp. 724–733.

[34] B. Milch, B. Marthi, S. Russell, D. Sontag, D.L. Ong, A. Kolobov, BLOG: Probabilistic models with unknown objects, in: L.P. Kaelbling, A. Saffiotti (Eds.), Proceedings of the 19th International Joint Conference on Artificial Intelligence, 2005, pp. 1352–1359.

[35] B. Milch, S. Russell, First-order probabilistic languages: Into the unknown, in: S. Muggleton, R. Otero, A. Tamaddoni-Nezhad (Eds.), Proceedings of the 16th International Conference on Inductive Logic Programming, in: LNAI, vol. 4455, 2007, pp. 10–24.

[36] S. Muggleton, Stochastic logic programs, in: L. De Raedt (Ed.), Advances in Inductive Logic Programming, IOS Press, 1996, pp. 254–264.

[37] R.E. Neapolitan, Learning Bayesian Networks, Prentice Hall, 2004.

[38] K.S. Ng, Learning comprehensible theories from structured data, PhD thesis, Computer Sciences Laboratory, The Australian National University, 2005.

[39] N.J. Nilsson, Probabilistic logic, Artificial Intelligence 28 (1) (1986) 71–88.

[40] R.T. Ng, V.S. Subrahmanian, Probabilistic logic programming, Information and Computation 101 (2) (1992) 150–201.

[41] D. Poole, Logic, knowledge representation, and Bayesian decision theory, in: Proceedings of the 1st International Conference on Computational Logic LNCS, vol. 1861, 2000, pp. 70–86.

[42] S. Park, F. Pfenning, S. Thrun, A probabilistic language based upon sampling functions, in: Conference Record of the 32nd Annual ACM Symposium on Principles of Programming Languages, 2005.

[43] M. Richardson, P. Domingos, Markov logic networks, Machine Learning 62 (2006) 107–136.

[44] N. Ramsey, A. Pfeffer, Stochastic lambda calculus and monads of probability distributions, in: Conference Record of the 29th Annual ACM Symposium on Principles of Programming Languages, 2002.

[45] A. Shirazi, E. Amir, Probabilistic modal logic, in: R.C. Holte, A. Howe (Eds.), Proceedings of the 22nd AAAI Conference on Artificial Intelligence, 2007, pp. 489–495.

[46] S. Shapiro, Classical logic II—Higher-order logic, in: L. Goble (Ed.), The Blackwell Guide to Philosophical Logic, Blackwell, 2001, pp. 33–54.

[47] S. Thompson, Type Theory and Functional Programming, Addison-Wesley, 1991.

[48] J. van Benthem, K. Doets, Higher-order logic, in: D.M. Gabbay, F. Guenther (Eds.), Handbook of Philosophical Logic, vol. 1, Reidel, 1983, pp. 275–330.

[49] J. Williamson, Probability logic, in: D. Gabbay, R. Johnson, H.J. Ohlbach, J. Woods (Eds.), Handbook of the Logic of Inference and Argument: The Turn Toward the Practical, in: Studies in Logic and Practical Reasoning, vol. 1, Elsevier, 2002, pp. 397–424.