DOI:10.1145/2699411

Open-universe probability models show merit in unifying efforts.

BY STUART RUSSELL

Unifying Logic and Probability

PERHAPS THE MOST enduring idea from the early days of AI is that of a *declarative* system reasoning over explicitly represented knowledge with a general inference engine. Such systems require a formal language to describe the real world; and *the real world has things in it*. For this reason, classical AI adopted first-order logic—the mathematics of objects and relations—as its foundation.

The key benefit of first-order logic is its expressive power, which leads to concise—and hence learnable—models. For example, the rules of chess occupy 10^0 pages in first-order logic, 10^5 pages in propositional logic, and 10^{38} pages in the language of finite automata. The power comes from separating predicates from their arguments and quantifying over those arguments: so one can write rules about $On(p, c, x, y, t)$ (piece p of color c is on square x, y at move t) without filling in each specific value for c, p, x, y, and t.

Modern AI research has addressed another important property of the real world—*pervasive uncertainty* about both its state and its dynamics—using probability theory. A key step was Pearl's development of *Bayesian networks*, which provided the beginnings of a formal language for probability models and enabled rapid progress in reasoning, learning, vision, and language understanding. The expressive power of Bayes nets is, however, limited. They assume a fixed set of *variables*, each taking a value from a fixed *range*; thus, they are a *propositional* formalism, like Boolean circuits. The rules of chess and of many other domains are beyond them.

What happened next, of course, is that classical AI researchers noticed the pervasive uncertainty, while modern AI researchers noticed, or remembered, that the world has things in it. Both traditions arrived at the same place: *the world is uncertain* **and** *it has things in it*. To deal with this, we have to *unify logic and probability*.

But how? Even the meaning of such a goal is unclear. Early attempts by Leibniz, Bernoulli, De Morgan, Boole, Peirce, Keynes, and Carnap (surveyed by Hailperin[12] and Howson[14]) involved attaching probabilities to logical sentences. This line of work influenced AI

» key insights

- **First-order logic and probability theory have addressed complementary aspects of knowledge representation and reasoning: the ability to describe complex domains concisely in terms of objects and relations and the ability to handle uncertain information. Their unification holds enormous promise for AI.**
- **New languages for defining open-universe probability models appear to provide the desired unification in a natural way. As a bonus, they support probabilistic reasoning about the existence and identity of objects, which is important for any system trying to understand the world through perceptual or textual inputs.**

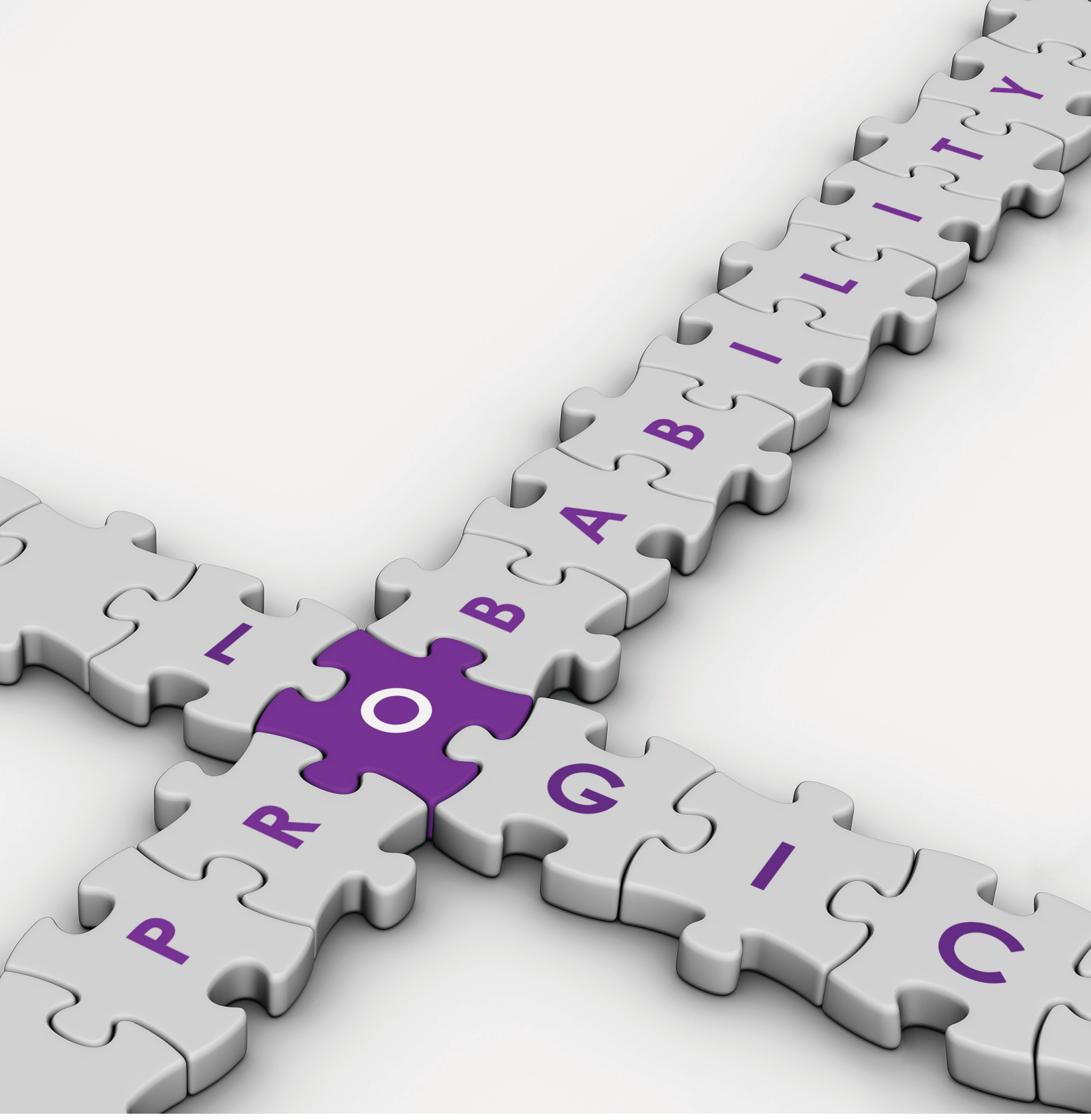


IMAGE BY ALMAGAMI

research but has serious shortcomings as a vehicle for representing knowledge.

An alternative approach, arising from both branches of AI and from statistics, combines the syntactic and semantic devices of logic (composable function symbols, logical variables, quantifiers) with the compositional semantics of Bayes nets. The resulting languages enable the construction of very large probability models and have changed the way in which real-world data are analyzed.

Despite their successes, these approaches miss an important consequence of uncertainty in a world of things: *uncertainty about what things are in the world*. Real objects seldom wear unique identifiers or preannounce their existence like the cast of a play. In areas such as vision, language understanding, Web mining, and computer security, the existence of objects must be *inferred* from raw data (pixels, strings, and so on) that contain no explicit object references.

The difference between knowing all the objects in advance and inferring their existence from observation corresponds to the distinction between *closed-universe* languages such as SQL and logic programs and *open-universe* languages such as full first-order logic. This article focuses in particular on open-universe probability models. The section on Open-Universe Models describes a formal language, Bayesian logic or BLOG, for writing such models.[21]

It gives several examples, including (in simplified form) a global seismic monitoring system for the Comprehensive Nuclear Test-Ban Treaty.

Logic and Probability

This section explains the core concepts of logic and probability, beginning with *possible worlds*.[a] A possible world is a formal object (think "data structure") with respect to which the truth of any assertion can be evaluated.

For the language of propositional logic, in which sentences are composed from proposition symbols $X_1, \ldots, X_n$ joined by logical connectives ($\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\Leftrightarrow$), the possible worlds $\omega \in \Omega$ are all possible assignments of *true* and *false* to the symbols. First-order logic adds the notion of *terms*, that is, expressions referring to objects; a term is a constant symbol, a logical variable, or a k-ary function applied to k terms as arguments. Proposition symbols are replaced by atomic sentences, consisting of either predicate symbols applied to terms or equality between terms. Thus, *Parent*(*Bill*, *William*) and *Father*(*William*) = *Bill* are atomic sentences. The quantifiers $\forall$ and $\exists$ make assertions across all objects, for example,

$$\forall p, c \;\; (Parent(p, c) \wedge Male(p)) \Leftrightarrow Father(c) = p.$$

For first-order logic, a possible world specifies (1) a set of domain elements (or objects) $o_1, o_2, \ldots$ and (2) mappings from the constant symbols to the domain elements and from the function and predicate symbols to functions and relations on the domain elements. Figure 1a shows a simple example with two constants and one binary predicate. Notice that first-order logic is an open-universe language: even though there are two constant symbols, the possible worlds allow for 1, 2, or indeed arbitrarily many objects. A closed-universe language enforces additional assumptions:

- The *unique names* assumption requires that distinct terms must refer to distinct objects.
- The *domain closure* assumption requires that there are no objects other than those named by terms.

These two assumptions force every world to contain the same objects, which are in one-to-one correspondence with the ground terms of the language (see Figure 1b).[b] Obviously, the set of worlds under open-universe semantics is larger and more heterogeneous, which makes the task of defining open-universe probability models more challenging.

The formal semantics of a logical language define the truth value of a sentence in a possible world. For example, the first-order sentence $A = B$ is true in world ω iff A and B refer to the same object in ω; thus, it is true in the first three worlds of Figure 1a and false in the fourth. (It is *always* false under closed-universe semantics.) Let $T(\alpha)$ be the set of worlds where the sentence α is true; then one sentence α *entails* another sentence β, written $\alpha \models \beta$, if $T(\alpha) \subseteq T(\beta)$. Logical inference algorithms generally determine whether a *query* sentence is entailed by the known sentences.

In probability theory, a *probability model* P for a countable space Ω of possible worlds assigns a probability $P(\omega)$ to each world, such that $0 \leq P(\omega) \leq 1$ and $\Sigma_{\omega \in \Omega} P(\omega) = 1$. Given a probability model, the probability of a logical sentence α is the total probability of all worlds in which α is true:

$$P(\alpha) = \Sigma_{\omega \in T(\alpha)} P(\omega). \qquad (1)$$

The *conditional* probability of one sentence given another is $P(\alpha | \beta) = P(\alpha \wedge \beta) / P(\beta)$, provided $P(\beta) > 0$. A *random variable* is a function from possible worlds to a fixed range of values; for example, one might define the Boolean random variable $V_{A=B}$ to have the value *true* in the first three worlds of Figure 1a and *false* in the fourth. The *distribution* of a random variable is the set of probabilities associated with each of its possible values. For example, suppose the variable *Coin* has values 0 (heads) and 1 (tails). Then the assertion *Coin* $\sim$ *Bernoulli*(0.6) says that *Coin* has a Bernoulli distribution with parameter 0.6, that is, a probability 0.6 for the value 1 and 0.4 for the value 0.

a In logic, a possible world may be called a *model* or *structure*; in probability theory, a *sample point*. To avoid confusion, this paper uses "model" to refer only to probability models.

b The open/closed distinction can also be illustrated with a common-sense example. Suppose a system knows that *Father*(*William*) = *Bill* and *Father*(*Junior*) = *Bill*. How many children does Bill have? Under closed-universe semantics—for example, in a database—he has exactly two; under open-universe semantics, between 1 and ∞.

Figure 1. (a) Some of the infinitely many possible worlds for a first-order, open-universe language with two constant symbols, *A* and *B*, and one binary predicate *R*(*x*, *y*). Gray arrows indicate the interpretations of *A* and *B* and black arrows connect pairs of objects satisfying *R*. (b) The analogous figure under closed-universe semantics; here, there are exactly four possible *x*, *y*-pairs and hence $2^4 = 16$ worlds.

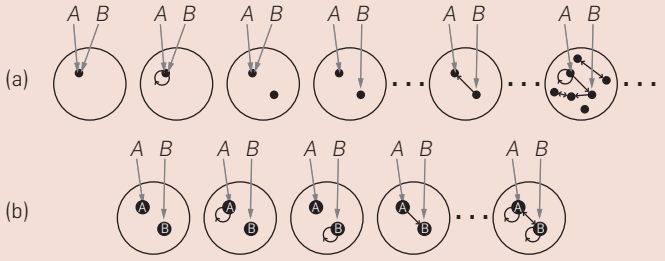
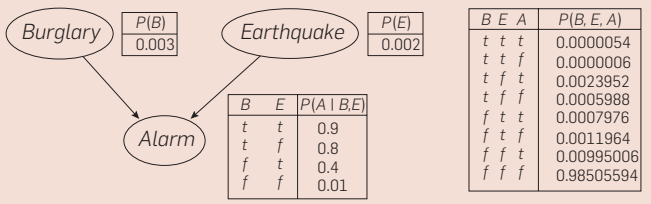


Figure 2. Left: a Bayes net with three Boolean variables, showing the conditional probability of *true* for each variable given its parents. Right: the joint distribution defined by Equation (2).



Unlike logic, probability theory lacks broad agreement on syntactic forms for expressing nontrivial assertions. For communication among statisticians, a combination of English and LaTeX usually suffices; but precise language definitions are needed as the "input format" for general probabilistic reasoning systems and as the "output format" for general learning systems. As noted, *Bayesian networks*[27] provided a (partial) syntax and semantics for the propositional case. The syntax of a Bayes net for random variables $X_1, \ldots, X_n$ consists of a directed, acyclic graph whose nodes correspond to the random variables, together with associated local conditional distributions.[c] The semantics specifies a joint distribution over the variables as follows:

$$P(\omega) = P(x_1, \ldots, x_n) = \prod_i P(x_i | \text{parents}(X_i)). \quad (2)$$

These definitions have the desirable property that *every well-formed Bayes net corresponds to a proper probability model on the associated Cartesian product space*. Moreover, a sparse graph—reflecting a sparse causal structure in the underlying domain—leads to a representation that is exponentially smaller than the corresponding complete enumeration.

The Bayes net example in Figure 2 (due to Pearl) shows two independent causes, *Earthquake* and *Burglary*, that influence whether or not an *Alarm* sounds in Professor Pearl's house. According to Equation (2), the joint probability $P(Burglary, Earthquake, Alarm)$ is given by

$$P(Burglary)\, P(Earthquake)\, P(Alarm \mid Burglary, Earthquake).$$

The results of this calculation are shown in the figure. Notice the eight possible worlds are the same ones that would exist for a propositional logic theory with the same symbols.

A Bayes net is more than just a specification for a distribution over worlds; it is also a stochastic "machine" for *generating* worlds. By sampling the variables in topological order (that is, parents before children), one generates a world exactly according to the distribution defined in Equation (2). This generative view will be helpful in extending Bayes nets to the first-order case.

Adding Probabilities to Logic

Early attempts to unify logic and probability attached probabilities directly to logical sentences. The first rigorous treatment, Gaifman's propositional *probability logic*,[9] was augmented with algorithmic analysis by Hailperin[12] and Nilsson.[23] In such a logic, one can assert, for example,

$$P(Burglary \Rightarrow Earthquake) = 0.997006, \quad (3)$$

a claim that is implicit in the model of Figure 2. The sentence $Burglary \Rightarrow Earthquake$ is true in six of the eight possible worlds; so, by Equation (1), assertion (3) is equivalent to

$$P(ttt) + P(ttf) + P(ftt) + P(ftf) + P(fft) + P(fff) = 0.997006.$$

Because any particular probability model μ assigns a probability to every possible world, such a constraint will be either true or false in μ; thus, μ, a distribution over possible propositional worlds, acts as a single possible world, with respect to which the truth of any probability assertion can be evaluated. Entailment between probability assertions is then defined in exactly the same way as in ordinary logic; thus, assertion (3) entails the assertion

$$P(Burglary \wedge Earthquake) \leq 0.997006,$$

because the latter is true in every probability model in which assertion (3) holds. Satisfiability of sets of such assertions can be determined by linear programming.[12] Hence, we have a "probability logic" in the same sense as "temporal logic"—that is, a deductive logical system specialized for reasoning with probabilistic assertions.

To apply probability logic to tasks such as proving the theorems of probability theory, a more expressive language was needed. Gaifman[8] proposed a *first-order* probability logic, with possible worlds being first-order model structures and with probabilities attached to sentences of (function-free) first-order logic.

Within AI, the most direct descendant of these ideas appears in Lukasiewicz's *probabilistic logic programs*, in which a probability range is attached to each first-order Horn clause and inference is performed by solving linear programs, as suggested by Hailperin. Within the subfield of *probabilistic databases* one also finds logical sentences labeled with probabilities[6]—but in this case probabilities are attached directly to the tuples of the database. (In AI and statistics, probability is attached to general relationships whereas observations are viewed as incontrovertible evidence.) Although probabilistic databases can model complex dependencies, in practice one often finds such systems using global independence assumptions across tuples.

Halpern[13] and Bacchus[3] adopted and extended Gaifman's technical approach, adding *probability expressions* to the logic. Thus, one can write

$$\forall h_1, h_2\ Burglary(h_1) \wedge \neg\, Burglary(h_2) \Rightarrow P(Alarm(h_1)) > P(Alarm(h_2)),$$

where now *Burglary* and *Alarm* are predicates applying to individual houses. The new language is more expressive but does not resolve the difficulty that Gaifman faced—how to define complete and consistent probability models. Each inequality *constrains* the underlying probability model to lie in a half-space in the high-dimensional space of probability models. Conjoining assertions corresponds to intersecting the constraints. Ensuring that the intersection yields a single point is not easy. In fact, Gaifman's principal result[8] is a single probability model requiring (1) a probability for every possible ground sentence and (2) probability constraints for infinitely many existentially quantified sentences.

Researchers have explored two solutions to this problem. The first involves writing a partial theory and then "completing" it by picking out one canonical model in the allowed set. Nilsson[23] proposed choosing the *maximum entropy* model consistent with the specified constraints. Paskin[24] developed a "maximum-entropy probabilistic logic" with constraints expressed as weights (relative probabilities) attached to first-order clauses. Such models are often called *Markov logic networks* or MLNs[30] and have become a popular technique for applications involving relational data. There is, however, a semantic difficulty with such models: weights trained in one scenario do not generalize to scenarios with different numbers of objects.

c Texts on Bayes nets typically do not define a syntax for local conditional distributions other than tables, although Bayes net software packages do.

Moreover, by adding irrelevant objects to a scenario, one can change the model's predictions for a given query to an arbitrary extent.[16,20]

A second approach, which happens to avoid the problems just noted,[d] builds on the fact that every well-formed Bayes net necessarily defines a unique probability distribution—a *complete theory* in the terminology of probability logics—over the variables it contains. The next section describes how this property can be combined with the expressive power of first-order logical notation.

Bayes Nets with Quantifiers

Soon after their introduction, researchers developing Bayes nets for applications came up against the limitations of a propositional language. For example, suppose that in Figure 2 there are many houses in the same general area as Professor Pearl's: each one needs an *Alarm* variable and a *Burglary* variable with the same CPTs and connected to the *Earthquake* variable in the same way. In a propositional language, this *repeated structure* has to be built manually, one variable at a time. The same problem arises with models for sequential data such as text and time series, which contain sequences of identical submodels, and in models for Bayesian parameter learning, where every instance variable is influenced by the parameter variables in the same way.

At first, researchers simply wrote *programs* to build the networks, using ordinary loops to handle repeated structure. Figure 3 shows pseudocode to build an alarm network for R geological fault regions, each with $H(r)$ houses.

The pictorial notation of *plates* was developed to denote repeated structure and software tools such as BUGS[10] and Microsoft's infer.net have facilitated a rapid expansion in applications of probabilistic methods. In all these tools, the model structure is built by a fixed program, so every possible world has the same random variables connected in the same way. Moreover, the code for constructing the models is not viewed as something that could be the output of a learning algorithm.

d Briefly, the problems are avoided by separating the generative models for object existence and object properties and relations and by allowing for unobserved objects.

An open-universe probability model (OUPM) defines a probability distribution over possible worlds that vary in the objects they contain and in the mapping from symbols to objects. Thus, OUPMs can handle data from sources that violate the closed-universe assumption.

Breese[4] proposed a more declarative approach reminiscent of Horn clauses. Other declarative languages included Poole's Independent Choice Logic, Sato's PRISM, Koller and Pfeffer's probabilistic relational models, and de Raedt's Bayesian Logic Programs. In all these cases, the head of each clause or *dependency statement* corresponds to a parameterized set of child random variables, with the parent variables being the corresponding ground instances of the literals in the body of the clause. For example, Equation (4) shows the dependency statements equivalent to the code fragment in Figure 3:

$$\begin{aligned} &Burglary(h) \sim Bernoulli(0.003) \\ &Earthquake(r) \sim Bernoulli(0.002) \\ &Alarm(h) \sim \\ &\quad CPT[\ldots](Earthquake \\ &\quad (Faultregion(\mathrm{h})), Burglary(\mathrm{h})) \end{aligned} \tag{4}$$

where *CPT* denotes a suitable conditional probability table indexed by the corresponding arguments. Here, h and r are *logical* variables ranging over houses and regions; they are implicitly universally quantified. *FaultRegion* is a function symbol connecting a house to its geological region. Together with a *relational skeleton* that enumerates the objects of each type and specifies the values of each function and relation, a set of dependency statements such as Equation (4) corresponds to an ordinary, albeit potentially very large, Bayes net. For example, if there are two houses in region A and three in region B, the corresponding Bayes net is the one in Figure 4.

Open-Universe Models

As noted earlier, closed-universe languages disallow uncertainty about what things are in the world; the existence and identity of all objects must be known in advance.

In contrast, an open-universe probability model (OUPM) defines a probability distribution over possible worlds that vary in the objects they contain and in the mapping from symbols to objects. Thus, OUPMs can handle data from sources (text, video, radar, intelligence reports, among others) that violate the closed-universe assumption. Given evidence, OUPMs *learn* about the objects the world contains.

Looking at Figure 1a, the first problem is how to ensure that the model specifies a

proper distribution over a heterogeneous, unbounded set of possible worlds. The key is to extend the generative view of Bayes nets from the propositional to the first-order, open-universe case:

- Bayes nets generate propositional worlds one event at a time; each event fixes the value of a variable.
- First-order, closed-universe models such as Equation (4) define generation steps for entire classes of events.
- First-order, open-universe models include generative steps that *add objects to the world* rather than just fixing their properties and relations.

Consider, for example, an open-universe version of the alarm model in Equation (4). If one suspects the existence of up to three geological fault regions, with equal probability, this can be expressed as a *number statement*:

$$\#\,Region \sim UniformInt\,(1, 3)\,. \quad (5)$$

For the sake of illustration, let us assume that the number of houses in a region r is drawn uniformly between 0 and 4:

$$\#\,House(FaultRegion = r) \sim UniformInt(0, 4)\,. \quad (6)$$

Here, *FaultRegion* is called an *origin function*, since it connects a house to the region from which it originates.

Together, the dependency statements (4) and the two number statements (5 and 6), along with the necessary type signatures, specify a complete distribution over all the possible worlds definable with this vocabulary. There are infinitely many such worlds, but, because the number statements are bounded, only finitely many—317,680,374 to be precise—have non-zero probability. Figure 5 shows an example of a particular world constructed from this model.

The BLOG language[21] provides a precise syntax, semantics, and inference capability for open-universe probability models composed of dependency and number statements. BLOG models can be arbitrarily complex, but they inherit the key declarative property of Bayes nets: *every well-formed BLOG model specifies a well-defined probability distribution over possible worlds.*

To make such a claim precise, one must define exactly what these worlds are and how the model assigns a probability to each. The definitions (given in full in Brian Milch's PhD thesis[20]) begin with the objects each world contains. In the standard semantics of typed first-order logic, objects are just numbered tokens with types. In BLOG, each object also has an *origin*, indicating how it was generated. (The reason for this slightly baroque construction will become clear shortly.) For number statements with no origin functions—for example, Equation (5)—the objects have an empty origin; for example, $\langle Region, , 2\rangle$ refers to the second region generated from that statement. For number statements with origin functions—for example, Equation (6)—each object records its origin; for example, $\langle House, \langle FaultRegion, \langle Region, , 2\rangle\rangle, 3\rangle$ is the third house in the second region.

The *number variables* of a BLOG model specify how many objects there are of each type with each possible origin; thus $\#House_{\langle FaultRegion,\langle Region, ,2\rangle\rangle}(\omega) = 4$ means that in world ω there are 4 houses in region 2. The *basic variables* determine the values of predicates and functions for all tuples of objects; thus, $Earthquake_{\langle Region, ,2\rangle}(\omega) = true$ means that in world ω there is an earthquake in region 2. A possible world is defined by the values of all the number variables and basic variables. A world may be generated from the model by sampling in topological order, for example, see Figure 5.

Figure 3. Illustrative pseudocode for building a version of the Bayes net in Figure 2 that handles *R* different regions with *H*(*r*) houses in each.

```
loop for r from 1 to R do
    add node Earthquake_r with no parents, prior 0.002
    loop for h from 1 to H(r) do
        add node Burglary_r,h with no parents, prior 0.003
        add node Alarm_r,h with parents Earthquake_r, Burglary_r,h
```

Figure 4. The Bayes net corresponding to Equation (4), given two houses in region *A* and three in *B*.

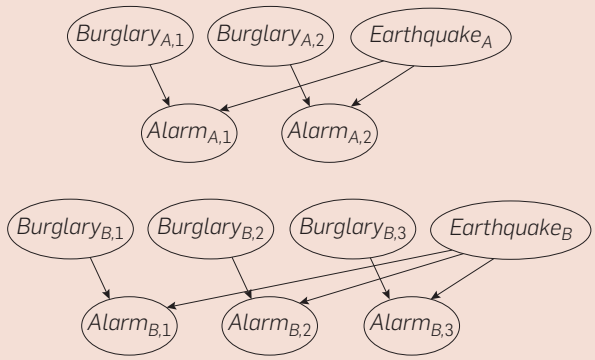


Figure 5. Construction of one possible world by sampling from the burglary/earthquake model. Each row shows the variable being sampled, the value it receives, and the probability of that value conditioned on the preceding assignments.

Variable	Value	Prob.
$\#Region$	2	0.3333
$Earthquake_{\langle Region, ,1\rangle}$	*false*	0.998
$Earthquake_{\langle Region, ,2\rangle}$	*false*	0.998
$\#House_{\langle FaultRegion,\langle Region, ,1\rangle\rangle}$	1	0.2
$\#House_{\langle FaultRegion,\langle Region, ,2\rangle\rangle}$	1	0.2
$Burglary_{\langle House,\langle FaultRegion,\langle Region, ,1\rangle\rangle,1\rangle}$	*false*	0.997
$Burglary_{\langle House,\langle FaultRegion,\langle Region, ,2\rangle\rangle,1\rangle}$	*true*	0.003

The probability of a world so constructed is the product of the probabilities for all the sampled values; in this case, 0.00003972063952. Now it becomes clear why each object contains its origin: this property ensures that every world can be constructed by exactly one sampling sequence. If this were not the case, the probability of a world would be the sum over all possible sampling sequences that create it.

Open-universe models may have infinitely many random variables, so the full theory involves nontrivial measure-theoretic considerations. For example, the number statement $\#Region \sim Poisson(\mu)$ assigns probability $e^{-\mu}\,\mu^k/k!$ to each nonnegative integer k. Moreover, the language allows recursion and infinite types (integers, strings, and so on). Finally, well-formedness disallows cyclic dependencies and infinitely receding ancestor chains; these conditions are undecidable in general, but certain syntactic sufficient conditions can be checked easily.

Examples

The standard "use case" for BLOG has three elements: the *model*, the *evidence* (the known facts in a given scenario), and the *query*, which may be any expression, possibly with free logical variables. The answer is a posterior joint probability for each possible set of substitutions for the free variables, given the evidence, according to the model.[e] Every model includes type declarations, type signatures for the predicates and functions, one or more number statements for each type, and one dependency statement for each predicate and function. (In the examples here, declarations and signatures are omitted where the meaning is clear.) Dependency statements use an if-then-else syntax to handle so-called *context-specific* dependencies, whereby one variable may or may not be a parent of another, depending on the value of a third variable.

e As with Prolog, there may be infinitely many sets of substitutions of unbounded size; designing exploratory interfaces for such answers is an interesting HCI challenge.

Figure 6. BLOG model for citation information extraction. For simplicity the model assumes one author per paper and omits details of the grammar and error models. OM(a, b) is a discrete log-normal, base 10, that is, the order of magnitude is $10^{\alpha\pm b}$.

```
type Researcher, Paper, Citation;
random String Name(Researcher);
random String Title(Paper);
random Paper PubCited(Citation);
random String Text(Citation);
random Boolean Prof(Researcher);
origin Researcher Author(Paper);
#Researcher ~ OM(3,1);
Name(r) ~ CensusDB_NamePrior();
Prof(r) ~ Boolean(0.2);
#Paper(Author=r)
   if Prof(r) then ~ OM(1.5,0.5) else ~ OM(1,0.5);
Title(p) ~ CSPaperDB_TitlePrior();
CitedPaper(c) ~ UniformChoice(Paper p);
Text(c) ~ HMMGrammar(Name(Author(CitedPaper(c))),
                     Title(CitedPaper(c)));
```

Figure 7. BLOG model for radar tracking of multiple targets. $X(a, t)$ is the state of aircraft a at time t, while $Z(b)$ is the observed position of blip b.

```
#Aircraft(EntryTime = t) ~ Poisson(λa);
Exits(a,t) if InFlight(a,t) then ~ Boolean(αe);
InFlight(a,t) =
   (t == EntryTime(a)) | (InFlight(a,t-1) & !Exits(a,t-1));
X(a,t) if t = EntryTime(a) then ~ InitState()
   else if InFlight(a,t) then ~ Normal( F*X(a,t-1),Σx);
#Blip(Source=a, Time=t)
  if InFlight(a,t) then ~ Bernoulli(DetectionProb(X(a,t)));
#Blip(Time=t) ~ Poisson(λf);
Z(b) if Source(b)=null then ~ UniformInRegion(R);
   else ~ Normal(H*X(Source(b),Time(b)),Σb);
```

Citation matching. Systems such as CiteSeer and Google Scholar extract a database-like representation, relating papers and researchers by authorship and citation links, from raw ASCII citation strings. These strings contain no object identifiers and include errors of syntax, spelling, punctuation, and content, which lead in turn to errors in the extracted databases. For example, in 2002, CiteSeer reported over 120 distinct books written by Russell and Norvig.

A generative model for this domain (Figure 6) connects an underlying, unobserved world to the observed strings: there are researchers, who have names; researchers write papers, which have titles; people cite the papers, combining the authors' names and the paper's title (with errors) into the text of the citation according to some grammar. Given citation strings as evidence, a multi-author version of this model, trained in an unsupervised fashion, had an error rate two to three times lower than CiteSeer's on four standard test sets.[25] The inference process in such a vertically integrated model also exhibits a form of collective, knowledge-driven disambiguation: the more citations there are for a given paper, the more accurately each of them is parsed, because the parses have to agree on the facts about the paper.

Multitarget tracking. Given a set of unlabeled data points generated by some unknown, time-varying set of objects, the goal is to detect and track the underlying objects. In radar systems, for example, each rotation of the radar dish produces a set of blips. New objects may appear, existing objects may disappear, and false alarms and detection failures are possible. The standard model (Figure 7) assumes independent, linear–Gaussian dynamics and measurements. Exact inference is provably intractable, but MCMC typically works well in practice. Perhaps more importantly, elaborations of the scenario (formation flying, objects heading for unknown destinations, objects taking off or landing) can be handled by small changes to the model without resorting to new mathematical derivations and complex programming.

Nuclear treaty monitoring. Verifying the Comprehensive Nuclear-Test-Ban Treaty requires finding all seismic events on Earth above a minimum magnitude. The UN CTBTO maintains a network of sensors, the International Monitoring System (IMS); its automated processing software, based on 100 years of seismology research, has a detection failure rate of about 30%. The NET-VISA system,[2] based on an OUPM, significantly reduces detection failures.

The NET-VISA model (Figure 8) expresses the relevant geophysics directly. It describes distributions over the number of events in a given time interval (most of which are naturally occurring) as well as over their time, magnitude, depth, and location. The locations of natural events are distributed according to a spatial prior trained (like other parts of the model) from historical data; man-made events are, by the treaty rules, assumed to occur uniformly. At every station s, each phase (seismic wave type) p from an event e produces either 0 or 1 detections (above-threshold signals); the detection probability depends on the event magnitude and depth and its distance from the station. ("False alarm" detections also occur according to a station-specific rate parameter.) The measured arrival time, amplitude, and other properties of a detection d depend on the properties of the originating event and its distance from the station.

Once trained, the model runs continuously. The evidence consists of detections (90% of which are false alarms) extracted from raw IMS waveform data and the query typically asks for the most likely event history, or *bulletin*, given the data. For reasons previously explained, NET-VISA uses a special-purpose inference algorithm. Results so far are encouraging: for example, in 2009 the UN's SEL3 automated bulletin missed 27.4% of the 27,294 events in the magnitude range 3–4 while NET-VISA missed 11.1%. Moreover, comparisons with dense regional networks show that NET-VISA finds up to 50% more real events than the final bulletins produced by the UN's expert seismic analysts. NET-VISA also tends to associate more detections with a given event, leading to more accurate location estimates (see Figure 9). The CTBTO has announced its intention to deploy NET-VISA as soon as possible.

Remarks on the examples. Despite superficial differences, the three examples are structurally similar: there are unknown objects (papers, aircraft, earthquakes) that generate percepts according to some physical process (citation, radar detection, seismic propagation). The same structure and reasoning patterns hold for areas such as database deduplication and natural language understanding. In some cases, inferring an object's existence involves grouping percepts together—a process that resembles the clustering task in machine learning. In other cases, an object may generate no percepts at all and still have its existence inferred—as happened, for example, when observations of Uranus led to the discovery of Neptune. Allowing object trajectories to be nonindependent in Figure 7 enables such inferences to take place.

Inference

By Equation (1), the probability $P(\alpha|\mathbf{e})$ for a closed query sentence α given evidence $\mathbf{e}$ is proportional to the sum of probabilities for all worlds in which α and $\mathbf{e}$ are satisfied, with the probability for each world being a product of model parameters as explained earlier.

Many algorithms exist to calculate or approximate this sum of products for Bayes nets. Hence, it is natural to consider *grounding* a first-order probability model by instantiating the

Figure 8. The NET-VISA model.

```
#SeismicEvents ~ Poisson(T*λe);
Natural(e) ~ Boolean(0.999);
Time(e) ~ UniformReal(0,T);
Magnitude(e) ~ Exponential(log(10));
Depth(e) if Natural(e) then ~ UniformReal(0,700) else = 0;
Location(e) if Natural(e) then ~ SpatialPrior()
   else ~ UniformEarthDistribution();
#Detections(event=e, phase=p, station=s)
  if IsDetected(e,p,s) then = 1 else = 0;
IsDetected(e,p,s) ~
   Logistic(weights(s,p), Magnitude(e), Depth(e), Distance(e,s));
#Detections(site = s) ~ Poisson(T*λf(s));
OnsetTime(d,s)
   if (event(d) = null) then ~ UniformReal(0,T)
   else = Time(event(d)) + Laplace(μt(s),σt(s)) +
     GeoTT(Distance(event(d),s),Depth(event(d)),phase(d));
Amplitude(d,s)
   if (event(d) = null) then ~ NoiseAmplitudeDistribution(s)
   else ~ AmplitudeModel(Magnitude(event(d)),
                Distance(event(d),s),Depth(event(d)),phase(d));
... and similar clauses for azimuth, slowness, and phase type.
```

Figure 9. Location estimates for the DPRK nuclear test of February 12, 2013: UN CTBTO Late Event Bulletin (green triangle); NET-VISA (blue square). The tunnel entrance (black cross) is 0.75 km from NET-VISA's estimate. Contours show NET-VISA's posterior location distribution.

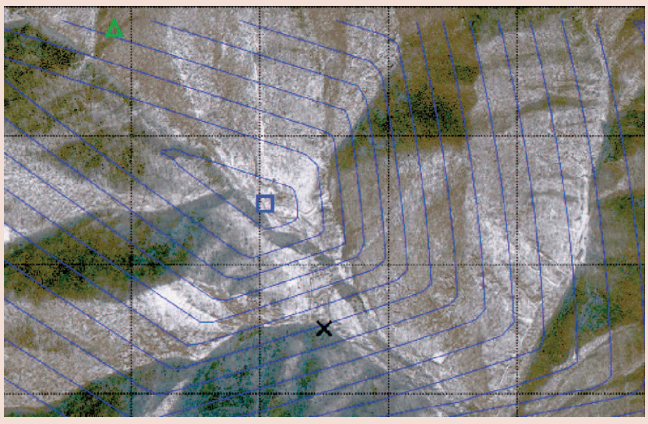


Figure 10. A CHURCH program that expresses the burglary/earthquake model in Equations (4)–(6).

```
(define num-regions (mem (lambda () (uniform 1 3))))
(define-record-type region (fields index))
(define regions (map (lambda (i) (make-region i))
                     (iota(num-regions)))
(define num-houses (mem (lambda (r)(uniform 0 4))))
(define-record-type house (fields fault-region index))
(define houses (map (lambda (r)
                      (map (lambda (i) (make-house r i))
                           (iota (num-houses r)))) regions)))
(define earthquake (mem (lambda (r) (flip 0.002))))
(define burglary (mem (lambda (h) (flip 0.003))))
(define alarm (mem (lambda (h)
  (if (burglary h)
      (if (earthquake (house-fault-region h))
          (flip 0.9) (flip 0.8))
      (if (earthquake (house-fault-region h))
          (flip 0.4) (flip 0.01))))
```

logical variables with "all possible" ground terms in order to generate a Bayes net, as illustrated in Figure 4; then, existing inference algorithms can be applied.

Because of the large size of these ground networks, exact inference is usually infeasible. The most general method for approximate inference is Markov chain Monte Carlo. MCMC methods execute a random walk among possible worlds, guided by the relative probabilities of the worlds visited and aggregating query results from each world. MCMC algorithms vary in the choice of neighborhood structure for the random walk and in the *proposal distribution* from which the next state is sampled; subject to an ergodicity condition on these choices, samples will converge to the true posterior in the limit. MCMC scales well with network size, but its *mixing time* (the time needed before samples reflect the true posterior) is sensitive to the quantitative structure of the conditional distributions; indeed, hard constraints may prevent convergence.

BUGS[10] and MLNs[30] apply MCMC to a preconstructed ground network, which requires a bound on the size of possible worlds and enforces a propositional "bit-vector" representation for them. An alternative approach is to apply MCMC directly on the space of first-order possible worlds[26, 31]; this provides much more freedom to use, for example, a sparse graph or even a relational database[19] to represent each world. Moreover, in this view it is easy to see that MCMC moves can not only alter relations and functions but also add or subtract objects and change the interpretations of constant symbols; thus, MCMC can move among the open-universe worlds shown in Figure 1a.

As noted, a typical BLOG model has infinitely many possible worlds, each of potentially infinite size. As an example, consider the multitarget tracking model in Figure 7: the function $X(a, t)$, denoting the state of aircraft a at time t, corresponds to an infinite sequence of variables for an unbounded number of aircraft at each step. Nonetheless, BLOG's MCMC inference algorithm converges to the correct answer, under the usual ergodicity conditions, for any well-formed BLOG model.[20]

The algorithm achieves this by sampling not completely specified possible worlds but *partial* worlds, each corresponding to a disjoint set of complete worlds. A partial world is a *minimal self-supporting instantiation*[f] of a subset of the *relevant* variables, that is, ancestors of the evidence and query variables. For example, variables $X(a, t)$ for values of t greater than the last observation time (or the query time, whichever is greater) are irrelevant so the algorithm can consider just a finite prefix of the infinite sequence. Moreover, the algorithm eliminates isomorphisms under object renumberings by computing the required combinatorial ratios for MCMC transitions between partial worlds of different sizes.[22]

MCMC algorithms for first-order languages also benefit from *locality* of computation[27]: the probability ratio between neighboring worlds depends on a subgraph of constant size around the variables whose values are changed. Moreover, a logical query can be evaluated *incrementally* in each world visited, usually in constant time per world, rather than recomputing it from scratch.[19, 31]

Despite these optimizations, generic inference for BLOG and other first-order languages remains too slow. Most real-world applications require a special-purpose proposal distribution to reduce the mixing time. A number of avenues are being pursued to resolve this issue:

- *Compiler techniques* can generate inference code that is specific to the model, query, and/or evidence. Microsoft's infer.net uses such methods to handle millions of variables. Experiments with BLOG show speedups of over 100×.[18]
- *Special-purpose probabilistic hardware*—such as Analog Devices' GP5 chip—offers further constant-factor speedups.
- *Special-purpose samplers* jointly sample groups of variables that are tightly constrained. A library of such samplers may render most user programs efficiently solvable.
- *Static analysis* can transform programs for efficiency[5, 15] and identify exactly solvable submodels.[7]

Finally, the rapidly advancing techniques of *lifted inference*[29] aim to unify probability and logic at the inferential level, borrowing and generalizing ideas from logical theorem proving. Such methods, surveyed by Van den Broeck,[32] may avoid grounding and take advantage of symmetries by manipulating symbolic distributions over large sets of objects.

Learning

Generative languages such as BLOG and BUGS naturally support Bayesian parameter learning with no modification: parameters are defined as random variables with priors and ordinary inference yields posterior parameter distributions given the evidence. For example, in Figure 8 we could add a prior $\lambda_e \sim \text{Gamma}(3, 0.1)$ for the seismic rate parameter λ_e instead of fixing its value in advance; learning proceeds as data arrive, even in the unsupervised case where no ground-truth events are supplied. A "trained

f An instantiation of a set of variables is self-supporting if the parents of every variable in the set are also in the set.

model" with fixed parameter values can be obtained by, for example, choosing maximum *a posteriori* values. In this way many standard machine learning methods can be implemented using just a few lines of modeling code. Maximum-likelihood parameter estimation could be added by stating that certain parameters are learnable, omitting their priors, and interleaving maximization steps with MCMC inference steps to obtain a stochastic online EM algorithm.

Structure learning—generating new dependencies and new predicates and functions—is more difficult. The standard idea of trading off degree of fit and model complexity can be applied, and some model search methods from the field of inductive logic programming can be generalized to the probabilistic case, but as yet little is known about how to make such methods computationally feasible.

Probability and Programs

Probabilistic programming languages or PPLs[17] represent a distinct but closely related approach for defining expressive probability models. The basic idea is that a randomized algorithm, written in an ordinary programming language, can be viewed not as a program to be executed, but as a probability model: a distribution over the possible *execution traces* of the program, given inputs. Evidence is asserted by fixing any aspect of the trace and a query is any predicate on traces. For example, one can write a simple Java program that rolls three six-sided dice; fix the sum to be 13; and ask for the probability that the second die is even. The answer is a sum over probabilities of complete traces; the probability of a trace is the product of the probabilities of the random choices made in that trace.

The first significant PPL was Pfeffer's IBAL,[28] a functional language equipped with an effective inference engine. CHURCH,[11] a PPL built on Scheme, generated interest in the cognitive science community as a way to model complex forms of learning; it also led to interesting connections to computability theory[1] and programming language research. Figure 10 shows the burglary/earthquake example in CHURCH; notice that the PPL code builds a possible-world data structure explicitly. Inference in CHURCH uses MCMC, where each move resamples one of the stochastic primitives involved in producing the current trace.

Execution traces of a randomized program may vary in the new objects they generate; thus, PPLs have an open-universe flavor. One can view BLOG as a declarative, relational PPL, but there is a significant semantic difference: in BLOG, in any given possible world, every ground term has a single value; thus, expressions such as $f(1)=f(1)$ are true by definition. In a PPL, on the other hand, $f(1)=f(1)$ may be false if f is a stochastic function, because each instance of $f(1)$ corresponds to a distinct piece of the execution trace. Memoizing every stochastic function (via `mem` in Figure 10) restores the standard semantics.

Prospects

These are early days in the process of unifying logic and probability. Experience in developing models for a wide range of applications will uncover new modeling idioms and lead to new kinds of programming constructs. And of course, inference and learning remain the major bottlenecks.

Historically, AI has suffered from insularity and fragmentation. Until the 1990s, it remained isolated from fields such as statistics and operations research, while its subfields—especially vision and robotics—went their own separate ways. The primary cause was *mathematical incompatibility*: what could a statistician of the 1960s, well-versed in linear regression and mixtures of Gaussians, offer an AI researcher building a robot to do the grocery shopping? Bayes nets have begun to reconnect AI to statistics, vision, and language research; first-order probabilistic languages, which have both Bayes nets and first-order logic as special cases, will extend and broaden this process.

Acknowledgments

My former students Brian Milch, Hanna Pasula, Nimar Arora, Erik Sudderth, Bhaskara Marthi, David Sontag, Daniel Ong, and Andrey Kolobov contributed to this research, as did NSF, DARPA, the Chaire Blaise Pascal, and ANR.

References

1. Ackerman, N., Freer, C., Roy, D. On the computability of conditional probability. arXiv 1005.3014, 2013.
2. Arora, N.S., Russell, S., Sudderth, E. NET-VISA: Network processing vertically integrated seismic analysis. *Bull. Seism. Soc. Am. 103* (2013).
3. Bacchus, F. *Representing and Reasoning with Probabilistic Knowledge*. MIT Press, 1990.
4. Breese, J.S. Construction of belief and decision networks. *Comput. Intell. 8* (1992) 624–647.
5. Claret, G., Rajamani, S.K., Nori, A.V., Gordon, A.D., Borgström, J. Bayesian inference using data flow analysis. In *FSE-13* (2013).
6. Dalvi, N.N., Ré, C., Suciu, D. Probabilistic databases. *CACM 52*, 7 (2009), 86–94.
7. Fischer, B., Schumann, J. AutoBayes: A system for generating data analysis programs from statistical models. *J. Funct. Program 13* (2003).
8. Gaifman, H. Concerning measures in first order calculi. *Israel J. Math. 2* (1964), 1–18.
9. Gaifman, H. Concerning measures on Boolean algebras. *Pacific J. Math. 14* (1964), 61–73.
10. Gilks, W.R., Thomas, A., Spiegelhalter, D.J. A language and program for complex Bayesian modelling. *The Statistician 43* (1994), 169–178.
11. Goodman, N.D., Mansinghka, V.K., Roy, D., Bonawitz, K., Tenenbaum, J.B. Church: A language for generative models. In *UAI-08* (2008).
12. Hailperin, T. Probability logic. *Notre Dame J. Formal Logic 25*, 3 (1984), 198–212.
13. Halpern, J.Y. An analysis of first-order logics of probability. *AIJ 46*, 3 (1990), 311–350.
14. Howson, C. Probability and logic. *J. Appl. Logic 1*, 3–4 (2003), 151–165.
15. Hur, C.-K., Nori, A.V., Rajamani, S.K., Samuel, S. Slicing probabilistic programs. In *PLDI-14* (2014).
16. Jain, D., Kirchlechner, B., Beetz, M. Extending Markov logic to model probability distributions in relational domains. In *KI-07* (2007).
17. Koller, D., McAllester, D.A., Pfeffer, A. Effective Bayesian inference for stochastic programs. In *AAAI-97* (1997).
18. Li, L., Wu, Y., Russell, S. SWIFT: Compiled inference for probabilistic programs. Tech. Report EECS-2015-12, UC Berkeley, 2015.
19. McCallum, A., Schultz, K., Singh, S. FACTORIE: Probabilistic programming via imperatively defined factor graphs. In *NIPS 22* (2010).
20. Milch, B. Probabilistic models with unknown objects. PhD thesis, UC Berkeley, 2006.
21. Milch, B., Marthi, B., Sontag, D., Russell, S.J., Ong, D., Kolobov, A. BLOG: Probabilistic models with unknown objects. In *IJCAI-05* (2005).
22. Milch, B., Russell, S.J. General-purpose MCMC inference over relational structures. In *UAI-06* (2006).
23. Nilsson, N.J. Probabilistic logic. *AIJ 28* (1986), 71–87.
24. Paskin, M. Maximum entropy probabilistic logic. Tech. Report UCB/CSD-01-1161, UC Berkeley, 2002.
25. Pasula, H., Marthi, B., Milch, B., Russell, S.J., Shpitser, I. Identity uncertainty and citation matching. In *NIPS 15* (2003).
26. Pasula, H., Russell, S.J. Approximate inference for first-order probabilistic languages. In *IJCAI-01* (2001).
27. Pearl, J. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.
28. Pfeffer, A. IBAL: A probabilistic rational programming language. In *IJCAI-01* (2001).
29. Poole, D. First-order probabilistic inference. In *IJCAI-03* (2003).
30. Richardson, M., Domingos, P. Markov logic networks. *Machine Learning 62*, 1–2 (2006), 107–136.
31. Russell, S.J. Expressive probability models in science. In *Discovery Science* (Tokyo, 1999).
32. Van den Broeck, G. *Lifted Inference and Learning in Statistical Relational Models*. PhD thesis, Katholieke Universiteit Leuven, 2013.

Stuart Russell (russell@cs.berkeley.edu), is a professor of computer science and Smith-Zadeh Professor of Engineering at the University of California, Berkeley.

Watch the authors discuss their work in this exclusive *Communications* video. http://cacm.acm.org/videos/unifying-logic-and-probability