Soft Constraints for Inference with Declarative Knowledge

Zenna Tavares [1] **Javier Burroni** [2] **Edgar Minaysan** [3] **Armando Solar Lezama** [1] **Rajesh Ranganath** [4]

Abstract

We develop a likelihood free inference procedure for conditioning a probabilistic model on a predicate. A predicate is a Boolean valued function which expresses a yes/no question about a domain. Our contribution, which we call predicate exchange, constructs a softened predicate which takes value in the unit interval [0, 1] as opposed to a simply true or false. Intuitively, 1 corresponds to true, and a high value (such as 0.999) corresponds to "nearly true" as determined by a distance metric. We define Boolean algebra for soft predicates, such that they can be negated, conjoined and disjoined arbitrarily. A softened predicate can serve as a tractable proxy to a likelihood function for approximate posterior inference. However, to target exact inference, we temper the relaxation by a temperature parameter, and add a accept/reject phase use to replica exchange Markov Chain Mont Carlo, which exchanges states between a sequence of models conditioned on predicates at varying temperatures. We describe a lightweight implementation of predicate exchange that it provides a language independent layer that can be implemented on top of existingn modeling formalisms.

1. Introduction

Conditioning in Bayesian inference incorporates observed data into a model. In a broader sense, conditioning revises a model such that a yes/no question (a predicate) is resolved to a true proposition (a fact). For instance, the question of whether a variable is equal to a particular value, changes from a predicate of uncertain truth, to a fact, once it is observed. In principle, a predicate can be used to declare any fact about a domain, not only the observation of data. In practice, sampling from models conditioned on most predicates presents severe challenges to existing inference procedures.

Predicates can be used to update a model to adhere to known facts about a domain, without the burden of specifying how to revise the model. For example, in inverse graphics (Marschner & Greenberg, 1998; Kulkarni et al., 2015) (inferring three dimensional geometry from observed images), the proposition "rigid bodies do not intersect" is a predicate on latent configurations of geometry. To manually revise a model to constructively adhere to this fact is ranges between inconvenient and infeasible. Instead, we would ideally simply condition on it being true, concentrating probability mass on physically plausible geometric configurations, ultimately to yield more accurate posterior inferences in the inverse graphics problem.

Predicates can also express observations that are more abstract than variables in a model. In diabetes research for example, probabilistic models have been used to relate physiological factors to glucose levels over time (Levine et al., 2017; Murata et al., 2004). Rather than concrete, numerical glucose measurements, a medical practitioner may observe (or be told) that a patient suffers from recurrent hypoglycemia, i.e., that their glucose levels periodically fall below a critical value. Even if the occurrence of hypoglycemia does not appear as an explicit variable in the model, it could be constructed as a predicate on glucose levels, and conditioned on to infer the posterior distribution over latent physiological factors.

Several effective sampling (Andrieu et al., 2003) and variational (Jordan et al., 1999; Ranganath et al., 2014) approaches to inference require only a black-box likelihood function, i.e., one evaluable on arbitrary input. The likelihood function quantifies the extent to which values of latent variables are consistent with observations. However, most models conditioned on most predicates have likelihood functions that are intractable to compute or unknown. For example, conditioning random variables that are deterministic transformations of other random variables (e.g., the presence of hypoglycemia in the example above, or the mean of a collection of variables) often results in likelihoods that are normalized by intractable integrals. In other cases, the likelihood function is implicit to a generative process, rather than explicitly specified, and hence unavailable even when the condition is a conventional observation.

[*]Equal contribution [1]MIT, USA [2]UMass Amherst, USA [3]Princeton University, USA [4]NYU, USA. Correspondence to: Zenna Tavares <zenna@mit.edu>.

In this paper we present predicate exchange: a likelihood-free method to sample from distributions conditioned on predicates from a broad class. It is composed of two parts:

1. **Predicate Relaxation** transforms a predicate such that it returns a value in a soft Boolean algebra: the unit interval $[0, 1]$ with continuous logical connectives $\tilde{\wedge}$. $\tilde{\vee}$ and $\neg$.

2. **Replica Exchange** simulates several Markov chains of a model at different temperatures. Temperature is a parameter of predicate relaxation which controls the amount of approximation it introduces. We adapt standard replica exchange to draw samples that are asymptotically exact from the unrelaxed model.

By returning a value in $[0, 1]$ instead of $\{0, 1\}$, a soft predicate quantifies the extent to which values of latent variables are consistent with the predicate. This allows it to serve a role similar to a likelihood function, and opens up the use of likelihood-based inference procedures. Orthogonally, we embed $]0, 1]$ in a Boolean algebra to support the expression of domain knowledge of composite Boolean structure. Continuing the previous example, we may know that a person does *not* have hypoglycemia, or that they have hypoglycemia *or* hyperglycemia, or *neither*.

Predicate exchange is motivated by probabilistic programming languages, which have vastly expanded the class of probabilistic models that can be expressed,, but still heavily restrict the kinds of predicates that can be conditioned on. Rather than introduce a new language or modeling formalism, we mirror (Wingate et al., 2011) and provide a light-weight implementation that performs inference by modulating the execution of a stochastic simulation based model. This means predicate exchange is easily incorporated into most frameworks.

Our approach comes with certain limitations. Equality conditions on continuous variables indicate sets of zero measure. This is problematic because the probability of proposing a satisfying state in a Markov chain becomes zero. In these cases predicate exchange must sample at a minimum temperature strictly greater than zero, which is approximate. Another limitation occurs if a predicate has branches (e.g., if-then-else statements) which depend on uncertainty in the model.

In summary we address the problem of conditioning probabilistic models on predicates as a means to express declarative knowledge. In detail, we:

1. Formalize simulation based probabilistic models in measure theoretic probability, and conditioning as the imposition of constraints expressed as predicates (Section 3).

2. Motivate predicate relaxation (Section 4.1), and provide a complete soft Boolean algebra.

3. Provide a light-weight implementation of predicate exchange (Section 5) through nonstandard execution of a simulation based model.

4. Evaluate our approach on examples, including a case study in glycemic forecasting.

2. Related Work

Demand for likelihood-free inference emerged in genetics ecology. Tavaré et al. (1997) compared summary statistics of the output of a simulation with that of observed data, and rejected mismatches. Weiss et al. (1998) expanded on this with a tolerance term, so that simulations yielding data sufficiently close to the targets were accepted. Approximate Bayesian Computation (ABC) has come to refer to broad class of methods (Beaumont et al., 2002; Sisson et al., 2007) in this general regime. Marjoram et al. (2003) simulated Markov Chains according to the prior, but introduced the accept/reject stage to yield approximate posterior samples. A small tolerance leads to a high rejection rate, whereas a large tolerance results in an unacceptable approximation error. Among several solutions are dynamically decreasing the tolerance (Toni et al., 2008), importance reweighting samples based on distance (Wegmann et al., 2009), adapting the tolerance based on distance (Del Moral et al., 2012; Lenormand et al., 2013), as well as annealing the tolerance as a temperature parameter (Albert et al., 2015).

Predicate exchange targets simulation models and uses distance metrics, but targets exact inference without summary statistics. A recent approach (Graham et al., 2017) with similar objectives develops a Hamiltonian Monte Carlo variant, using a quasi-Newton method during leap-frog integration to exactly solve the observation constraint. This is limited to differentiable models conditioned with equality.

Probabilistic logics such as ProbLog (Richardson & Domingos, 2006) and Markov logic networks (De Raedt et al., 2007) allow extend first order logic to declare both models and conditions. More recent probabilistic programming systems (Milch et al., 2007; Wood et al., 2014; Mansinghka et al., 2014; Goodman et al., 2008; Carpenter et al., 2017) have focused on stochastic simulation, and automatically automatically derive the likelihood function for a rich class of models.

Several continuous (Levin, 2000) and fuzzy (Klir & Yuan, 1995) logics apply model-theoretic tools to metric structures. Continuous logics replace the Boolean structure $\{T, F\}$, quantifiers $\forall x$ and $\exists x$, and logical connectives with continuous counter-parts. Predicate relies uses a continuous logic only make inference more tractable. Semantically,

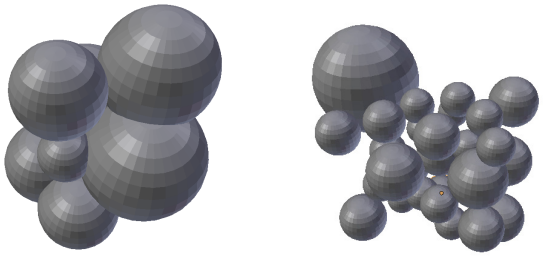

Figure 1. Sample from geometric prior (left), whereas (right) is conditioned on no-intersection constraint

our approach remains within measure theoretic foundations, which relies on hard predicates to condition.

3. Simulation Models

Probabilistic simulation based models specify the step-by-step causal mechanisms of a domain, and use probability distributions for any uncertain parameters. A simulation model can be stochastically executed, using a random number generator to sample from primitive random variables in the model. Inference means to simulate the model while imposing constraints on variables in the model. This is difficult, since simulation based models lack an explicit likelihood function, which is necessary for most inference procedures.

Conditioning on predicates requires a measure-theoretic foundation, in which a simulation model is a random variable:

Random Variables. Probability models lie on top of probability spaces. A probability space is a measure space $(\Omega, \mathcal{H}, \mathcal{P})$, where $\mathcal{H}$ is a sigma algebra and $\mathcal{P}(\Omega) = 1$ (Çınlar, 2011). Random variables are functions from the space Ω to a realization space $\mathcal{X}$. As a concrete example the space Ω can be thought of as a hypercube, with $\mathcal{P}$ being uniform over that hypercube. To build a normal random variable, we need a function that maps from $\Omega \to \mathbb{R}$. If the underlying probability space is uniform, then this function is the inverse cumulative distribution function of the normal.

A *model* $\mathcal{M}$ is a collection of random variables along with a probability space.

Conditioning Conditioning a model creates a new model. As an example consider a model $\mathcal{M}$ with two random variables X_1 and X_2 that both take real values. Conditioning $\mathcal{M}$ on $X_1 = 1$, defines a new model $\mathcal{M}_{|A}$ based on limiting the measure space Ω to the set $A = \{\omega : X_1(\omega) = 1\}$. The new model is defined on a new probability space

$$(\Omega \cap A, \{A \cap B, B \in \mathcal{H}\}, \mathcal{P}/\mathcal{P}(A)) \tag{1}$$

with the same random variables X_1 and X_2. Sampling from $\mathcal{M}_{|A}$ produces samples only where $X_1 = 1$

More generally, conditioning on any predicate $Y(\omega) = \ell(X_1(\omega), \ldots, X_n(\omega))$ defines a new model defined exactly as above, where $A = \{\omega : \ell(X_1(\omega), \ldots, X_n(\omega)) = 1\}$. Sampling from $\mathcal{M}_{|A}$ generates $(x_1, ..., x_n)$ where ℓ is true.

The general construction of new models might require conditioning on sets of measure zero. This process can be made rigorous via disintegration (Chang & Pollard, 1997). Disintegration can be thought of as the reversal of building joint distributions through product measure constructions.

4. Predicate Exchange

To condition a model $\mathcal{M}$ on a predicate Y we develop *predicate exchange*, a likelihood-free inference procedure. It is composed of two parts:

1. **Predicate Relaxation** constructs a relaxed predicate $\tilde{Y}$ from Y. $\tilde{Y}$ takes values in a soft Boolean algebra: the unit interval $[0, 1]$ with continuous logical connectives $\tilde{\wedge}$. $\tilde{\vee}$ and $\neg$.

   $\tilde{Y}$ is 1 iff Y is 1, but otherwise takes nonzero values denoting the degree to which Y is satisfied.

2. **Replica Exchange** is a Markov Chain Monte Carlo procedure that exploits temperature. The strength by which $\tilde{Y}$ relaxes Y is modulated by a temperature parameter α, which trades off between accuracy and ease of inference. By simulating several replicas of $\tilde{Y}$ at different temperatures, replica exchange is able to draw exact samples.

4.1. Predicate Relaxation

A relaxed predicate $\tilde{Y}$ approximates Y in the sense that when viewed as a likelihood function on model parameters, $\tilde{Y}$ has a broader support, assigning nonzero weights to parameter values which have zero weight under Y. There are three desiderata which govern this approximation. First, $\tilde{Y}$ should have a temperature parameter α that controls the fidelity of the approximation. In particular, $\tilde{Y}$ should converge to Y as $\alpha \to 0$, and to a flat surface as $\alpha \to \infty$. Second, the fidelity of the approximation should vary monotonically with temperature. Third, $\tilde{Y}$ should be consistent with Y on 1. That is $Y(\omega) = 1$ iff $\tilde{Y}(\omega) = 1$ at all temperatures.

Definition 1. A function $\tilde{Y} : \Omega \to [0, 1]$ parameterized by $\alpha \in [0, \infty)$ is a relaxation of a $Y : \Omega \to \{0, 1\}$ if:

(i) For all $\omega \in \Omega$, $\lim_{\alpha \to 0} \tilde{Y}(\omega; \alpha) = Y(\omega)$.

(ii) For all $\omega \in \Omega$, $\lim_{\alpha \to \infty} \tilde{Y}(\omega; \alpha) = 1$.

(iii) For all α, $\tilde{Y}(\omega;\alpha) = 1$ iff $Y(\omega) = 1$.

(iv) The entropy $H(\tilde{Y}(\omega;\alpha))$ (which characterizes the fidelity of the approximation) is an increasing function of α.[1]

The construction of $\tilde{Y}$ from Y (Section 5) substitutes primitive predicates (equality, inequalities and logical operators) in the model with soft predicates. Soft predicates rely on a notion of distance; the degree to which a predicate is satisfied is a measure of closeness in a metric space. For example if x and y are real values, then $x \mathrel{\tilde{=}} y$ is defined as $k_\alpha(\rho(x,y))$ where ρ is a distance function and k_α is a relaxation kernel paramterized by temperature α. In general, we use $\tilde{p}$ to denote a relaxation of a predicate p. A relaxation kernel maps distances to values in $[0,1]$, and ensures that $\tilde{Y}$ adheres to the outlined criteria. There are several possible kernels but we restrict our attention to the squared exponentional kernel:

$$k_\alpha(r) = \exp\left(-\frac{r^2}{\alpha}\right) \tag{2}$$

Distance The distance ρ depends on the realization spaces of random variables. For canonical spaces such as $\mathbb{R}$ and $\mathbb{N}$ we default to Euclidean distances. For example, $x \mathrel{\tilde{=}} y$ is then defined as $\exp(\|x-y\|/\alpha)$. For elements of composite structure taking with a product typem $x, y \in \mathbb{T}_1 \times \cdots \times \mathbb{T}_n$, by default ρ takes a mean $\rho(x,y) = (1/n)\sum_{i=1}^n \rho(x_i, y_i)$.

A soft inequality such as $x \mathrel{\tilde{>}} y$ is function of the amount by which x must be increased (or y decreased) until $x > y$ is true. This is the distance between x and the interval $[y,\infty]$, where the distance between a point and any interval $[a,b]$ is the smallest distance between x and any element in $[a,b]$, and therefore 0 if $x \in [a,b]$:

$$\rho(x,[a,b]) = \begin{cases} a-b & \text{if } x < a \\ x-b & \text{if } x < b \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

The *intrinsic* degree of satisfaction $\tilde{\ell}_{\text{inf}}\ (m)$ of a realization $m = (x_1, \cdots x_n)$ of a model is then the smallest distance between that realization and any point in the constraint set, i.e.,

$$\tilde{\ell}_{\text{inf}}\ (m) = \rho(m, A) \tag{4}$$

where $A = \{(x_1,\ldots,x_n) \mid \ell(x_1,\ldots,x_n) = 1\}$, and $\rho(x,A) = \inf\{\rho(x,a) \mid a \in A\}$ As shown in the Section 5, $\tilde{\ell}_{\text{inf}}$ can be difficult to compute.

[1] By compactness, it is integrable for all α, when Ω has finite dimension

$$\begin{aligned} x \mathrel{\tilde{=}} y &= k_\alpha(\rho(x,y)) \\ x \mathrel{\tilde{>}} y &= k_\alpha(\rho(x,[y,\infty])) \\ x \mathrel{\tilde{<}} y &= k_\alpha(\rho(y,[-\infty,x])) \\ a \mathrel{\tilde{\wedge}} b &= \max(a,b) \\ a \mathrel{\tilde{\vee}} b &= \min(a,b) \end{aligned}$$

Figure 2. Soft Primitive Predicates

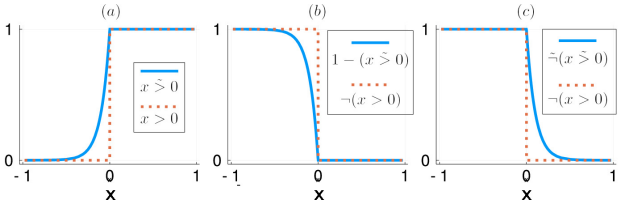

Figure 3. Soft predicates as function of x. In all figures the blue line denotes the soft predicate, while the red line denotes the predicate to approximate.

Negation Negation is a necessary component of a Boolean algebra, but its relaxation introduces complications. To illustrate, Figure 3 (a) shows $x \mathrel{\tilde{>}} 0$ as a function of x. In continuous logics (Kimmig et al., 2012), the negation of $a \in [0,1]$ is $1-a$. However, as shown in Figure 3 (b), this violates criteria (iii) of predicate relaxation; there are values which satisfy the hard predicate $\neg(x > 0)$ which do take a value of 1 in $1 - (x \mathrel{\tilde{>}} 0)$.

The problem of negation arises because $\tilde{Y}$ is consistent with Y at 1 but not at 0. In other words, $\tilde{Y}$ is a one-sided approximation. To overcome this challenge for models where negation is used, a two-sided relaxed predicate yields a pair (a_0, a_1), where $a_0, a_0 \in [0,1]$. a_1 preserves consistency with Y on 1, just as before, while a_0 preserves consistency with $\neg Y$ on 1 (or equivalently Y on 0). For example if $x \mathrel{\tilde{>}} 0 = (a_0, a_1)$, then as a function of x, a_0 and a_1 correspond to Figure 3 (a) and (c) respectively.

A complete two-sided soft logic is shown in Figure 4.1. Observe that soft negation simply swaps the elements of (a_0, a_1) to yield (a_1, a_0). Although a two-sided predicate has two components, for the sake of conditioning we are still concerned only with the true side a_1 in the pair (a_0, a_1).

Unsatisfiability Predicate exchange is unable to determine if a predicate is unsatisfiable (e.g. $(x > 1) \wedge (x < -1)$), and defers to the user to ensure this is the case.

$$x \tilde{=} y = (\text{if } x = y \text{ then } \alpha \text{ else } 1, k_\alpha(\rho(x, y)))$$
$$x \tilde{>} y = (k_\alpha(\rho(x, [-\infty, y])), k_\alpha(\rho(x, [y, \infty])))$$
$$x \tilde{<} y = (k_\alpha(\rho(y, [x, \infty])), k_\alpha(\rho(y, [-\infty, x])))$$
$$(a_0, a_1) \tilde{\wedge} (b_0, b_1) = (a_0 \tilde{\wedge} b_0, a_1 \tilde{\wedge} b_1)$$
$$(a_0, a_1) \tilde{\vee} (b_0, b_1) = (a_0 \tilde{\vee} b_0, a_1 \tilde{\vee} b_1)$$
$$\tilde{\neg} (a_0, a_1) = (a_1, a_0)$$

Figure 4. Two sided soft primitive predicates

4.2. Approximate Markov Chain Monte Carlo

A relaxed predicate can serve as an approximate likelihood, and as a result is amenable to likelihood based inference methods such as Markov Chain Monte Carlo. MCMC algorithms require a function f that is proportional to the the target density. In Bayesian inference this is the posterior, dictated by Bayes' theorem as the product of the likelihood and the prior. Approximate inference using relaxed predicates takes a similar form.

Let $\mathcal{M} = (X_1, \ldots, X_n)$ be a model, Y be a predicate that conditions $\mathcal{M}$, and $\tilde{Y}(\omega) = \tilde{\ell}(X_1(\omega), ..., X_n(\omega))$ be a relaxation of Y. Assuming a prior density p, the approximate posterior f is the product:

$$f(m) = p(m) \cdot \tilde{\ell}(m) \tag{5}$$

$\tilde{\ell}$ down weights parameter values which violate Y by the degree to which they violate it. This is modulated by the temperature α used in the relaxation kernels which constitute $\tilde{\ell}$. At maximum temperature $\tilde{\ell}$ has no effect, and the approximate posterior f is equal to the prior p. At zero temperature, f recovers the true posterior since parameter values which violate the condition are given zero weight.

For illustration, let $\mathcal{M} = (\mu, X)$ be a model where $\mu = \beta(3, 4), X = \mathcal{N}(\mu, 1)$ conditioned on $X_2 = 0.5$. The approximate posterior is shown at different temperatures in Figure 5 and defined as:

$$f_\alpha(\mu, x) = \beta_{0,1}(\mu) \cdot \mathcal{N}_{\mu,1}(x) \cdot k_\alpha(\rho(x, 0.5)) \tag{6}$$

The temperature parameter trades off between tractability of inference and the fidelity of the approximation. Too high and $\tilde{Y}$ will diverge too greatly from Y. Too low and convergence will be slow.

4.3. Replica Exchange

Replica exchange simulates (Swendsen & Wang, 1986) M replicas at different temperatures, and uses a Metropolis-Hastings update to periodically swap the temperatures

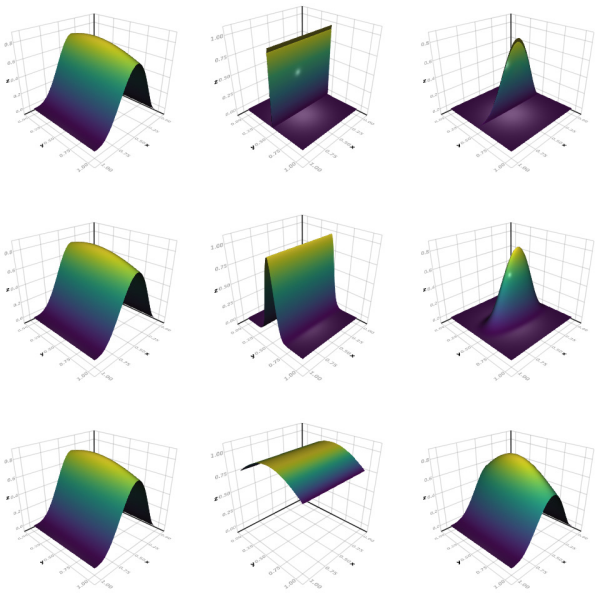

Figure 5. Approximate Posterior at varying temperatures. Temperature decreases from top row to bottom. Along each row: (left) is the prior term p, (center) is the soft likelihood term $\tilde{\ell}$, and (right) is the approximate posterior f

of chains. If f_{α_i} is an approximate posterior function at temperature α_i, two independent parallel chains simulating targets $f_{\alpha_1}(x)$, $f_{\alpha_2}(y)$ they follow a joint target $f_{\alpha_1,\alpha_2}(x, y) = f_{\alpha_1}(x) f_{\alpha_2}(y)$. Replica exchange swaps states between the chains while preserving the joint target. Swapping states is equivalent to swapping predicates, which motivates the name predicate exchange. Concretely, replica exchange proposes a swap from (x, y) to (y, x), and accepts it with probability $\min(1, A)$, where:

$$A = \frac{f_{\alpha_1,\alpha_2}(y, x)}{f_{\alpha_1,\alpha_2}(x, y)} = \frac{f_{\alpha_1}(y) f_{\alpha_2}(x)}{f_{\alpha_1}(x) f_{\alpha_2}(y)} \tag{7}$$

We modify standard replica exchange in two ways: (i) for exact inference, states which violate the constraint are rejected, and (ii) unlike conventional replica exchange which draws samples only from the zero-temperature chain, we accept states from any chain so long as $f_{\alpha_i}(x) = 1$.

Replica exchange has a number of hyper-parameters: the number of parallel chains, the corresponding temperatures, the swapping schedule. Several good practices are outlined in (Earl & Deem, 2005). In practice, we logarithmically space α between a lower and upper bound (e.g., $\log(\alpha_1) = 10^{-5}$, $\log(\alpha_M) = 10^5$), and swap states of chains that are adjacent in temperature (α_1 with α_2, α_2 with α_3, etc) periodically.

5. Implementation

In this section we describe a generic, lightweight implementation of predicate exchange. Our approach closely mirrors (Wingate et al., 2011; Milch et al., 2007) in the sense that it provides a language independent layer that can be implemented on top of existing programming languages and modeling formalisms. Our objective is to twofold: (i) to compute the prior term p, approximate likelihood term $\tilde{\ell}$, and approximate posterior term f (Equation 6) from an arbitrary program π, and (ii) to perform Replica Exchange MCMC to sample from this posterior.

A program π can be an arbitrary composition of deterministic and stochastic procedures, but all stochastic elements must come from a set of known *elementary random primitives*, or ERPs. ERPs correspond to primitive parametric distribution families, such as the uniform or normal distribution. Let $\mathcal{T}$ be a set of ERP types. Each type $\tau \in \mathcal{T}$ must support (i) evaluation of the conditional density $p_\tau(x \mid \theta_1, ..., \theta_n)$, and (ii) sampling from the distribution. Concretely, a conditioned program π is a any nullary program that contains the statements:

1. $\text{rand}(\tau, n, \theta_1, ..., \theta_n)$ returns a random sample from $p_\tau(x \mid \theta_1, ..., \theta_n)$. n is a unique named described below.
2. $\text{cond}(y)$ conditions π. It throws an error if $y \in \{0, 1\}$ is 0, and otherwise allows simulation to resume with no effect.

Example Program 1 illustrates a simple conditioned model.

5.1. Tracked Soft Execution

The prior term p is computed automatically as the product of random choices in the program. That is, let $\pi_{k|x_1,...,x_{k-1}}$ be the k'th ERP encountered in while executing π, x_k be the value it takes, and x denote the set of all values of all ERPs constructed in the simulation of π, $p(x)$ is the product:

$$p(x) = \prod_{k=1}^{K} p_\tau(x_k \mid \theta_1, ..., \theta_n) \tag{8}$$

Crucially, the parameters $\theta_1, .., \theta_n$ for each random variable may be fixed values or depend on values of other random variables in π.

Predicate exchange relies on softexecute (Algorithm 3), which formalizes the soft execution of a program π at temperature α, in the context of dictionary $\mathbb{D}$. $\mathbb{D}$ is a mutable mapping from a set of names to values. In the context of a particular dictionary, the simulation of a program is deterministic. This allows the simulation of π to be modulated by controlling the elements of $\mathbb{D}$.

Example Program 1

$x = \text{rand}(\mathcal{N}, x, 0, 1)$
$y = \text{rand}(\mathcal{N}, y, 0, 1)$
$\text{cond}(x > y)$
Return: (x, y)

Example Program 2

$x = \text{rand}(\mathcal{N}, x, 0, 1)$
if $x > 0$ **then**
  $\text{cond}(x = 1)$
else
  $\text{cond}(x = -100)$
end if
Return: x

softexecute simulates π but within a context where (i) variables $\ell_{\mathbb{D}}$ and $p_{\mathbb{D}}$ accumulate prior and approximate posterior values, and (ii) the following operators are redefined:

1. $\text{rand}(\tau, n, \theta_1, ...\theta_n)$ returns $\mathbb{D}(n)$, and in compliance with Equation 8 updates $p_{\mathbb{D}}$ with the conditional density. If n is not a key in $\mathbb{D}$, the distribution is sampled from and $\mathbb{D}(n)$ is updated with this value.
2. a op b and op a for $\text{op} \in \{>, <, =, \wedge, \vee, \neg\}$ are replaced with the softened counter-parts $\tilde{\text{op}} \in \{\tilde{>}, \tilde{<}, \tilde{=}, \tilde{\wedge}, \tilde{\vee}, \tilde{\neg}\}$.
3. $\text{cond}(y)$ updates $\tilde{\ell}_{\mathbb{D}}$ with $\tilde{\ell}_{\mathbb{D}} \;\tilde{\wedge}\; y$. $y \in [0, 1]$ due to soft primitive operators.

softexecute returns a real value for the approximate posterior of f as a function of the dictionary $\mathbb{D}$.

Control Flow Programs may have control flow constructs, such as if-then-else statements. These may cause softexecute to return a value that is significantly less than $\tilde{\ell}_{\text{inf}}$. This is because if a branch condition is a function of an uncertain value, then several unexplored alternative paths could produce values that are closer to the constraint set. softexecute is ignorant of thees other possibilities For illustration, consider Example Program 2. If $x = -1$ the condition fails, and the predicate relaxation will yield $x \;\tilde{=}\; -100$, which is significantly larger than if the true branch were taken.

Problems of this form appear in all forms of program analysis. This problem is called the path explosion problem, since the number of possible paths often increases combinatorially with program size and runtime length. Automated program testing, which is concerned with finding program paths that yield to failure has developed various strategies (Cadar et al., 2008; Sen et al., 2005). Unlike automated

Algorithm 3 Soft Execution: softexecute$(\pi, \alpha, \mathbb{D})$

Input: program π, temperature α, dictionary $\mathbb{D}$
Initialize $\tilde{\ell}_{\mathbb{D}} = 1, p_{\mathbb{D}} = 1$
Simulate π with following subroutines redefined as:
subroutine rand$(\tau, n, \theta_1, ..., \theta_n)$
  if $n \in \mathbb{D}$ **then**
    $x = \mathbb{D}(n)$
  else
    $x =$ sample from $p_\tau(x \mid \theta_1, ..., \theta_n)$
    Update dictionary: $\mathbb{D}(n) = x$
  end if
  $p_{\mathbb{D}} = p_{\mathbb{D}} \cdot p_\tau(x \mid \theta_1, ..., \theta_m)$
  Return from subroutine: x
end subroutine

subroutine cond(ℓ')
  $\tilde{\ell}_{\mathbb{D}} = \tilde{\ell}_{\mathbb{D}} \cdot \tilde{\ell}'_{\mathbb{D}}$
end subroutine

subroutine op$(x, \dots)$ for op $\in \{>, <, =, \wedge, \vee, \neg\}$
  Return from subroutine: $\tilde{\text{op}}\ (x, \dots)$
end subroutine

Return: $p_{\mathbb{D}} \cdot \tilde{\ell}_{\mathbb{D}}$

Algorithm 4 Predicate Exchange

Input: program π, temperatures $\alpha_1, ..., \alpha_m$, nsamples n
Input: mcmc, nsamples between swaps q
Initialize $\mathcal{D} =$ empty collection of dictionarys
Initialize $\mathbb{D}_1^{\text{init}}, ..., \mathbb{D}_m^{\text{init}}$ empty dictionarys
Define $f_{\alpha_i}(\mathbb{D}) =$ softexecute$(\pi, \alpha_i, \mathbb{D})$
repeat
  for $i = 1$ **to** m **do**
    $\mathbb{D}_1, ..., \mathbb{D}_q = q$ mcmc samples at temp α_i, from $\mathbb{D}_i^{\text{init}}$
    $\mathbb{D}_i^{\text{init}} = \mathbb{D}_q$
    for $j = 1$ **to** q **do**
      if $f_{\alpha_1}(\mathbb{D}_j) = 1$ **then**
        append $\mathbb{D}_j$ to $\mathcal{D}$
      end if
    end for
  end for
  for $i = m$ **down to** 2 **do**
    $j = i - 1$
    $p = f_{\alpha_i}(\mathbb{D}_j) f_{\alpha_j}(\mathbb{D}_i) / f_{\alpha_i}(\mathbb{D}_i) f_{\alpha_j}(\mathbb{D}_j)$
    if $p >$ random sample in $[0, 1]$ **then**
      swap α_i with α_j
    end if
  end for
until $\mathcal{D}$ has n elements
Return: $\mathcal{D}$

testing, probabilistic inference has the stricter requirement of adhering to the true posterior distribution. However, in predicate exchange, we have a latitude on all nonunitary values. This opens up the potential for extending program analysis methods to the probabilistic domain in future work.

5.2. Replica Exchange

Predicate exchange (Algorithm 4) performs replica exchange using softexectute as an approximate posterior. It takes as input an mcmc algorithm, which simulates an Markov Chain by manipulating elements of the $\mathbb{D}$. In our experiments, for finite dimensional continuous models we use the No U-Turn Sampler (Hoffman & Gelman, 2014), a variant of Hamiltonion Monte Carlo. We use reverse-mode automatic differentiation (Griewank & Walther, 2008) to compute the negative log gradient of f. For other models we use standard Metropolis Hastings by defining proposals on elements in the dictionary. In particular we use the single site MH (Wingate et al., 2011) which modifies a single random variable at a time.

6. Experiments

Small Models In Figure 7 we demonstrate two examples of conditioning on predicates which are non trivial. First we show that the conditioning can be used to truncate a Gaussian distribution, and the approximation behavior at varying temperatures. Second we show that two independent random variables can be made equal. While simple, both are a challenge for probabilistic programming systems because they prevent automatic calculation of the likelihood.

Glucose Model Type 2 diabetes is a prevalent and costly condition. Keeping blood glucose within normal limits helps prevent the long-term complications of Type 2 diabetes like diabetic neuropathy and diabetic retinopathy (Brownlee & Hirsch, 2006). Models to predict the trajectories of blood glucose aid in keeping glucose within normal limits (Zeevi et al., 2015). Traditional models have been built from compositions of differential equations (Albers et al., 2017; Levine et al., 2017) whose parameters are estimated separately for each patient. An alternative approach would be to use a flexible sequence model like an RNN. The problem with this approach is that an RNN can extrapolate to glucose values incompatible with human physiology. This is especially a problem where we have patients with only a few blood glucose measurements. To build an RNN model that respects physiology, we condition on it.

We compare the independent RNN model to the one with declarative knowledge on a second patient from Physionet (Moody et al., 2001). Figure 8 plots the results performed on more than 300 pairs of patients. We see that the conditional model simulates more realistic glucose dynamics for the

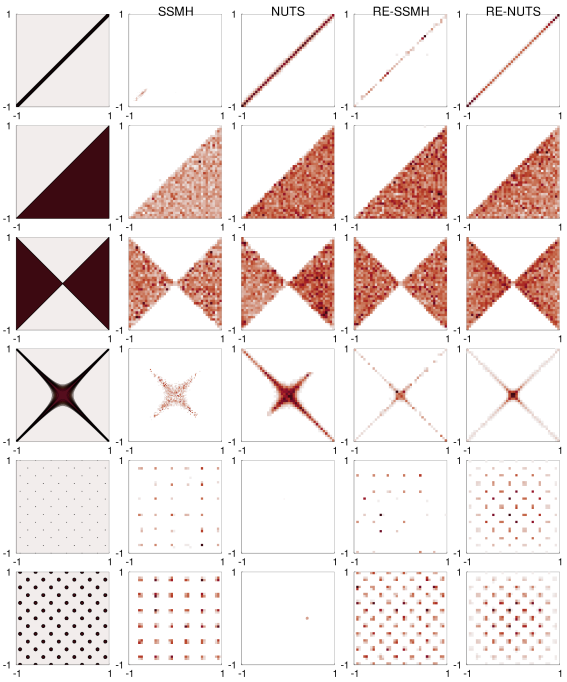

Figure 6. Posterior samples on different problems. RE: replica exchange, SSMH: Single Site Metropolis Hastings, HMC: Hamiltonian Monte Carlo. (Left) is target density, and following elements are histograms from computed samples.

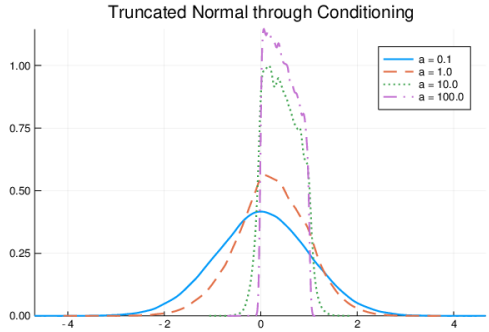

Figure 7. Left: Density from samples of Gaussian truncated to $[0, 1]$ through conditioning. Right: Conditioning on $X = Y$ where X and Y are independent normal distributions; shown at different temperatures.

patient with only a short observed time-series.

References

Albers, D. J., Levine, M., Gluckman, B., Ginsberg, H., Hripcsak, G., and Mamykina, L. Personalized glucose forecasting for type 2 diabetes using data assimilation. *PLoS computational biology*, 13(4):e1005232, 2017.

Albert, C., Künsch, H. R., and Scheidegger, A. A simulated annealing approach to approximate bayes computations.

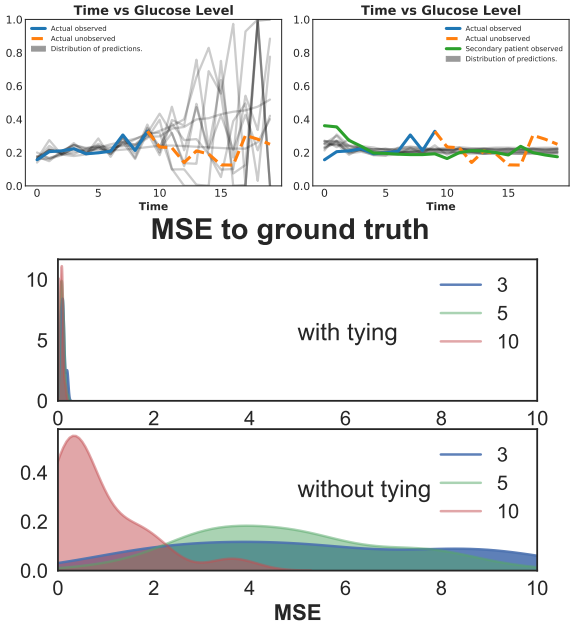

Figure 8. Left: Actual (dotted) and predicted trajectories that were learned using a partial trajectory. Center: Distribution of predicted trajectories learned only using the first ten data points and a tie with a secondary patient. Right, top: MSE when tie is present. Right, bottom: without tie. Tying expectations has dramatic influence on prediction error, while as more data is observed, the effect of tying decreases.

Statistics and computing, 25(6):1217–1232, 2015.

Andrieu, C., De Freitas, N., Doucet, A., and Jordan, M. I. An introduction to mcmc for machine learning. *Machine learning*, 50(1-2):5–43, 2003.

Beaumont, M. A., Zhang, W., and Balding, D. J. Approximate bayesian computation in population genetics. *Genetics*, 162(4):2025–2035, 2002.

Brownlee, M. and Hirsch, I. B. Glycemic variability: a hemoglobin a1c–independent risk factor for diabetic complications. *Jama*, 295(14):1707–1708, 2006.

Cadar, C., Ganesh, V., Pawlowski, P. M., Dill, D. L., and Engler, D. R. Exe: automatically generating inputs of death. *ACM Transactions on Information and System Security (TISSEC)*, 12(2):10, 2008.

Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., and Riddell, A. Stan: A probabilistic programming language. *Journal of statistical software*, 76(1), 2017.

Chang, J. T. and Pollard, D. Conditioning as disintegration. *Statistica Neerlandica*, 51(3):287–317, 1997.

Çınlar, E. *Probability and stochastics*, volume 261. Springer Science & Business Media, 2011.

De Raedt, L., Kimmig, A., and Toivonen, H. Problog: A probabilistic prolog and its application in link discovery. 2007.

Del Moral, P., Doucet, A., and Jasra, A. An adaptive sequential monte carlo method for approximate bayesian computation. *Statistics and Computing*, 22(5):1009–1020, 2012.

Earl, D. J. and Deem, M. W. Parallel tempering: Theory, applications, and new perspectives. *Physical Chemistry Chemical Physics*, 7(23):3910–3916, 2005.

Goodman, N. D., Mansinghka, V. K., Roy, D., Bonawitz, K., and Tenenbaum, J. B. Church: a language for generative models. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, pp. 220–229. AUAI Press, 2008.

Graham, M. M., Storkey, A. J., et al. Asymptotically exact inference in differentiable generative models. *Electronic Journal of Statistics*, 11(2):5105–5164, 2017.

Griewank, A. and Walther, A. *Evaluating derivatives: principles and techniques of algorithmic differentiation*, volume 105. Siam, 2008.

Hoffman, M. D. and Gelman, A. The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research*, 15(1):1593–1623, 2014.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

Kimmig, A., Bach, S., Broecheler, M., Huang, B., and Getoor, L. A short introduction to probabilistic soft logic. In *Proceedings of the NIPS Workshop on Probabilistic Programming: Foundations and Applications*, pp. 1–4, 2012.

Klir, G. and Yuan, B. *Fuzzy sets and fuzzy logic*, volume 4. Prentice hall New Jersey, 1995.

Kulkarni, T. D., Whitney, W. F., Kohli, P., and Tenenbaum, J. Deep convolutional inverse graphics network. In *Advances in neural information processing systems*, pp. 2539–2547, 2015.

Lenormand, M., Jabot, F., and Deffuant, G. Adaptive approximate bayesian computation for complex models. *Computational Statistics*, 28(6):2777–2796, 2013.

Levin, V. Basic concepts of continuous logics. *Kybernetes*, 29(9/10):1234–1249, 2000.

Levine, M. E., Hripcsak, G., Mamykina, L., Stuart, A., and Albers, D. J. Offline and online data assimilation for real-time blood glucose forecasting in type 2 diabetes. *arXiv preprint arXiv:1709.00163*, 2017.

Mansinghka, V., Selsam, D., and Perov, Y. Venture: a higher-order probabilistic programming platform with programmable inference. *arXiv preprint arXiv:1404.0099*, 2014.

Marjoram, P., Molitor, J., Plagnol, V., and Tavaré, S. Markov chain monte carlo without likelihoods. *Proceedings of the National Academy of Sciences*, 100(26):15324–15328, 2003.

Marschner, S. R. and Greenberg, D. P. *Inverse rendering for computer graphics*. Citeseer, 1998.

Milch, B., Marthi, B., Russell, S., Sontag, D., Ong, D. L., and Kolobov, A. 1 blog: Probabilistic models with unknown objects. *Statistical relational learning*, pp. 373, 2007.

Moody, G. B., Mark, R. G., and Goldberger, A. L. Physionet: a web-based resource for the study of physiologic signals. *IEEE Engineering in Medicine and Biology Magazine*, 20(3):70–75, 2001.

Murata, G. H., Hoffman, R. M., Shah, J. H., Wendel, C. S., and Duckworth, W. C. A probabilistic model for predicting hypoglycemia in type 2 diabetes mellitus: The diabetes outcomes in veterans study (doves). *Archives of internal medicine*, 164(13):1445–1450, 2004.

Ranganath, R., Gerrish, S., and Blei, D. Black box variational inference. In *Artificial Intelligence and Statistics*, pp. 814–822, 2014.

Richardson, M. and Domingos, P. Markov logic networks. *Machine learning*, 62(1-2):107–136, 2006.

Sen, K., Marinov, D., and Agha, G. Cute: a concolic unit testing engine for c. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pp. 263–272. ACM, 2005.

Sisson, S. A., Fan, Y., and Tanaka, M. M. Sequential monte carlo without likelihoods. *Proceedings of the National Academy of Sciences*, 104(6):1760–1765, 2007.

Swendsen, R. H. and Wang, J.-S. Replica monte carlo simulation of spin-glasses. *Physical review letters*, 57 (21):2607, 1986.

Tavaré, S., Balding, D. J., Griffiths, R. C., and Donnelly, P. Inferring coalescence times from dna sequence data. *Genetics*, 145(2):505–518, 1997.

Toni, T., Welch, D., Strelkowa, N., Ipsen, A., and Stumpf, M. P. Approximate bayesian computation scheme for parameter inference and model selection in dynamical systems. *Journal of the Royal Society Interface*, 6(31): 187–202, 2008.

Wegmann, D., Leuenberger, C., and Excoffier, L. Efficient approximate bayesian computation coupled with markov chain monte carlo without likelihood. *Genetics*, 2009.

Weiss, G. and von Haeseler, A. Inference of population history using a likelihood approach. *Genetics*, 149(3): 1539–1546, 1998.

Wingate, D., Stuhlmüller, A., and Goodman, N. Lightweight implementations of probabilistic programming languages via transformational compilation. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 770–778, 2011.

Wood, F., Meent, J. W., and Mansinghka, V. A new approach to probabilistic programming inference. In *Artificial Intelligence and Statistics*, pp. 1024–1032, 2014.

Zeevi, D., Korem, T., Zmora, N., Israeli, D., Rothschild, D., Weinberger, A., Ben-Yacov, O., Lador, D., Avnit-Sagi, T., Lotan-Pompan, M., et al. Personalized nutrition by prediction of glycemic responses. *Cell*, 163(5):1079–1094, 2015.