ENS DE LYON

UNIVERSITY OF OXFORD — DEPARTMENT OF COMPUTER SCIENCE

Internship report – First year of master's degree

May 9 – July 29, 2016

Variational Inference in Probabilistic Programs

a formal derivation of a Black-Box approach

Author:
Raphaël Monat
ENS Lyon
raphael.monat@ens-lyon.org

Supervisor:
Hongseok Yang
Professor of Computer Science, Oxford University
hongseok00X@gmail.com

Abstract

Probabilistic models are used in many fields to tackle different problems, ranging from image recognition to diagnosing diseases. The advantage of using models is that we can split the encoding of our problem into a probabilistic model from the ways we solve it. We can also classify models to develop some class-specific, but not problem-specific algorithms to solve given tasks. These algorithms are called inference algorithms. These classes of models ease the process of solving tasks, but scientists still need to develop class-specific algorithms. A new approach, at the intersection of Programming Languages and Machine Learning, is called Probabilistic Programming. One of the goals of this approach is to let the computer do the inference automatically, so we do not have to develop class-specific inference algorithms to accomplish our tasks. Probabilistic Programming Languages usually extend probabilistic models. However, researchers still need to derive general inference algorithms for probabilistic programs. In this report, we give a short introduction to probabilistic models and probabilistic programming. We encode probabilistic programs into a probabilistic transition system, and transform our inference task into an optimisation problem. We simplify this optimisation problem to derive a new, general, variational inference algorithm for probabilistic programs.

1 Introduction

Using probability to represent uncertainty, probabilistic models are able to treat noisy data measurements, and are very popular in a lot of scientific domains these days. For example, most voice recognition software use Hidden Markov Models [GY07]. These Hidden Markov Models are a class of probabilistic models. In medicine, Bayesian Networks are a good tool to diagnose patients by expressing relations between symptoms and diseases.

Bayesian Networks are a combination of directed acyclic graphs and probabilities, which express dependence between events. Each node represents an event, and the probability of such an event depends conditionally on its parent nodes. We present here a classical example of a Bayesian Network in Fig. 1. We will develop this example in the next sections, but we give some intuition in this introduction. This Bayesian Network represents the fact that the grass can be wet because of the sprinkler, or because of the rain, and that these last two events depend on whether the sky is cloudy. Probabilistic models give hypothesis on the shape of the data. Here, one hypothesis is that the grass can only be wet because it is raining, or because the sprinkler is on, so there is a dependency between the events "the grass is wet", "it is raining", "the sprinkler is on".

Given some data, we can perform queries on a probabilistic model to understand what we have learned from our observations. This querying is usually called inference. Inference aims at predicting hidden parameters or partially known data. To do inference, we will mainly use two probability rules, called the Bayes rule and the sum rule, presented in Eq. (1).

$$\Pr(A|B) = \frac{\Pr(B|A)\Pr(A)}{\Pr(B)} \qquad \Pr(A) = \sum_B \Pr(A, B) \tag{1}$$

An example of inference on Fig. 1 is: given the fact that the grass is wet and that the sprinkler is on, what is the probability that it is raining? That is, we are searching for $\Pr(R = T|W = T, S = T)$. Using the model in Fig.1, we can prove that $\Pr(R = T|W = T, S = T) \simeq 28.98\%$.

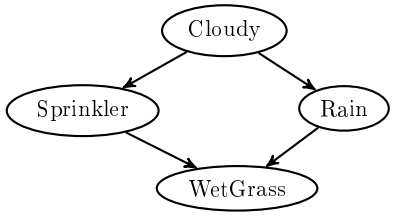
As we will see, computing by hand $\Pr(R = T|W = T, S = T)$ takes some time, even if the Bayes Network is quite simple. In fact, we can prove that the inference problem on Bayes Networks is NP-Complete [Coo90]. We can develop different algorithms to do exact inference on Bayes Networks, to avoid doing the inference by hand. It saves time, but if we had used another class of model, such as a Markov Network, we would have to derive other inference algorithms, and this is still costly.

The goal of probabilistic programming is to make life of scientists easier by letting the interpreter of a probabilistic programming language do the inference. Probabilistic programming is a new way to express probabilistic models, and it is able to express every other probabilistic model. It usually adds new constructions to programs, such as the ability to create new probability distributions, to sample from a distribution, to observe an event (sometimes called: to condition on an event), or to return the probability of a given event. Anglican, Church, and Venture [WvdMM14, GMR+12, MSP14] are examples of probabilistic programming languages. One real advantage of probabilistic programs is that scientists can focus on the design of their model rather than spending time developing new inference algorithms for new models, as inference can be done automatically by the computer. An example of a probabilistic program written in Anglican is presented in Listing. 1: this shows that describing probabilistic models and then querying the model is not difficult to encode in a probabilistic programming language. Research in probabilistic programming focuses on creating new and efficient inference algorithms, analysing a probabilistic program to know which inference algorithm to use, or rewriting programs to make inference easier.

There are two big families of inference strategies: one called sampling, and the other one called variational inference. Sampling consists in picking events according to the probability distribution

$\Pr(C = T)$	$\Pr(C = F)$
0.4	0.6

C	$\Pr(S = T)$	$\Pr(S = F)$
T	0.1	0.9
F	0.5	0.5



C	$\Pr(R = T)$	$\Pr(R = F)$
T	0.8	0.2
F	0.2	0.8

S	R	$\Pr(W = T)$	$\Pr(W = F)$
T	T	0.99	0.01
T	F	0.9	0.1
F	T	0.9	0.1
F	F	0	1

Figure 1: A simple bayes network

given by the model (for example, "it is cloudy, the sprinkler is off, it is raining, but the grass is not wet"), and then seeing if they satisfy the constraints in the conditional probability, and counting them accordingly. This strategy has the advantage of being asymptotically exact, but convergence can be slow. On the other hand, variational inference is usually faster, but more difficult to derive by hand. Variational inference consists in transforming the search for a conditional probability into an optimisation problem, and then using gradient descent (or other techniques) to solve this optimisation problem.

My internship was supervised by Hongseok Yang, Professor in the Department of Computer Science at the University of Oxford. It is part of my first year of Master's Degree at the ENS of Lyon. It lasted from May 9th, to July 29th. During my internship, I first started by reading a lot on variational inference as well as probabilistic programming. Then we searched for new inference algorithms based on variational inference. We formalized probabilistic programs into a probabilistic transition system, and then expressed the meaning of variational inference in this setting. I also regularly attended reading groups on Bayesian Machine Learning in the Department of Statistics and on Probabilistic Programming in the Department of Information Engineering.

Outline In the next section, we detail what a probabilistic model is, before giving more details on what probabilistic programming is and how inference on probabilistic programs works. Then, we present our variational inference for probabilistic programs, as well as some implementation details. Finally we give a brief presentation of the related work on probabilistic programming and on variational inference techniques.

2 Inference on probabilistic models

In this section, we first define probabilistic models. We show how we can query these models – that is, we present the concept of inference. Finally, we present exact and approximate methods used to perform inference.

Listing 1: Encoding of the Bayes Network in Anglican a Probabilistic Programming Language

```
(defquery bayes-net [sprinkler wet-grass]
  (let [is-cloudy (sample (flip 0.4))
        is-raining (cond (= is-cloudy true )
                         (sample (flip 0.8))
                         (= is-cloudy false)
                         (sample (flip 0.2)))
        sprinkler-dist (cond (= is-cloudy true)
                             (flip 0.1)
                             (= is-cloudy false)
                             (flip 0.5))
        wet-grass-dist (cond
                         (and (= sprinkler true)
                              (= is-raining true))
                         (flip 0.99)
                         (and (= sprinkler false)
                              (= is-raining false))
                         (false)
                         (or  (= sprinkler true)
                              (= is-raining true))
                         (flip 0.9))]
    (observe sprinkler-dist sprinkler)
    (observe wet-grass-dist wet-grass)
    (predict :is-raining is-raining)))

(def samples (doquery :pimh bayes-net [true true] :
   number-of-particles 10))

(frequencies (take 10000 samples))
```

2.1 What is a probabilistic model?

As we have briefly seen before, probabilistic models give hypothesis on the shape of the data we want to model. In the example shown in the introduction, we suppose that the grass can be wet because it rains, or because the sprinkler is on; the last two conditions themselves depending on whether the sky is cloudy or not.

There are different classes of probabilistic models, such as Bayesian Networks or Markov Models. Probabilistic models are usually defined in terms of graphs, because it enables the user to express models quite compactly, as opposed to describing the full joint probability for example.

We give a definition of a Bayesian Network, used in our example in Fig. 1:

Definition 1 (Bayesian Network)**.** A Bayesian Network consists of:

- A directed, acyclic graph $G = (V, E)$,
- A family of random variables indexed by the nodes of G, $(X_v)_{v \in V}$,
- A factorisation of the joint density probability giving a dependency relation: $\Pr(X) = \prod_{v \in V} \Pr(X_v | X_{pa(v)})$, where $pa(v) = \{x \in V \mid (x, v) \in E\}$ is the set of parents of v,
- A set of conditional probabilities: $\Pr(X_v = x_v \mid X_w = x_w \forall w \in Pred(v))$.

In our example, the set of conditional probabilities is given in the tables next to the nodes in Fig. 1, and the factorisation is expressed as:

$$\Pr(W, S, R, C) = \Pr(W|S, R) \Pr(S|C) \Pr(R|C) \Pr(C)$$

Using this rule and the sum rule, we can search for the global probability that the grass is wet:

$$\begin{aligned}
\Pr(W = T) &= \sum_{s,r,c \in \{T,F\}^3} \Pr(W = T, S = s, R = r, C = c) \\
&= \sum_{s,r,c \in \{T,F\}^3} \Pr(W = T|S = s, R = c) \Pr(S = s|C = c) \Pr(R = r|C = c) \Pr(C = c) \\
&= 65.988\%
\end{aligned}$$

We can see that the graphical representation of the Bayesian Network is much more compact than specifying the full joint probability. A table describing explicitly the joint probability $\Pr(W = w, S = s, R = r, C = c)$ for $(c, s, r, w) \in \{T, F\}^4$ would have size 16 in our particular example.

2.2 The inference problem

Using the factorisation of a Bayesian Network, we see that we can compute basic probabilities, but nothing with a conditional probability yet. However, it would be interesting to "query" the model, to know for example the probability that it is raining given the fact that the grass is wet and the sprinkler is on. This querying is called inference, and is computable using the Bayes rule and the sum rule (given in Eq. (1)). We prove that $\Pr(R = T|W = T, S = T) \simeq 28.98\%$ in Fig. 2.

$$\begin{aligned}\Pr(R=T|W=T,S=T) &= \frac{\Pr(R=T,W=T,S=T)}{\Pr(W=T,S=T)} \\ &= \frac{\sum_{c\in\{T,F\}}\Pr(W=T,S=T,R=T,C=c)}{\sum_{r,c\in\{T,F\}^2}\Pr(W=T,S=T,R=r,C=c)} \\ &= \frac{0.03168+0.0594}{0.03168+0.0594+0.0072+0.216} \simeq 28.98\%\end{aligned}$$

Because:

$$\begin{aligned}&\Pr(W=T,S=T,R=T,C=T) \\ &= \Pr(W=T|S=T,R=T)\Pr(S=T|C=T)\Pr(R=T|C=T)\Pr(C=T) \\ &= 0.99\cdot 0.1\cdot 0.8\cdot 0.4 = 0.03168 \\ &\Pr(W=T,S=T,R=T,C=F) \\ &= \Pr(W=T|S=T,R=T)\Pr(S=T|C=F)\Pr(R=T|C=F)\Pr(C=F) \\ &= 0.99\cdot 0.5\cdot 0.2\cdot 0.6 = 0.0594 \\ &\Pr(W=T,S=T,R=F,C=T) \\ &= \Pr(W=T|S=T,R=F)\Pr(S=T|C=T)\Pr(R=F|C=T)\Pr(C=T) \\ &= 0.9\cdot 0.1\cdot 0.2\cdot 0.4 = 0.0072 \\ &\Pr(W=T,S=T,R=F,C=F) \\ &= \Pr(W=T|S=T,R=F)\Pr(S=T|C=F)\Pr(R=F|C=F)\Pr(C=F) \\ &= 0.9\cdot 0.5\cdot 0.8\cdot 0.6 = 0.216\end{aligned}$$

Figure 2: Solving a query on a simple Bayes Network

$\Pr(R=T|W=T,S=T)$ is called the posterior of $R=T$ given the data $W=T,S=T$. In general, inference problems can be formulated as below:

$$\Pr(\theta|D,\mathcal{H}) = \frac{\Pr(D|\theta,\mathcal{H})\Pr(\theta|\mathcal{H})}{\Pr(D|\mathcal{H})}$$

- $\Pr(\theta|D,\mathcal{H})$ is called the posterior of θ given the data D (and the model $\mathcal{H}$),
- $\Pr(D|\theta,\mathcal{H})$ is the likelihood of parameters θ in the model $\mathcal{H}$,
- $\Pr(\theta|\mathcal{H})$ is called the prior probability of θ,
- $\Pr(D|\mathcal{H})$ is called the marginal likelihood. It is also seen as a normalisation term: sometimes, the relation above is written as $\Pr(\theta|D,\mathcal{H}) \propto \Pr(D|\theta,\mathcal{H})\Pr(\theta|\mathcal{H})$.

In our example, D was $W=T,S=T$, and θ was $R=T$. Here, $\mathcal{H}$ is a hidden parameter designing our Bayesian Model.

2.3 Exact methods to do inference

Performing inference over a class is usually a difficult problem. In fact, Gregory Cooper proved in [Coo90] that inference on a Bayesian Network is NP-Complete.

There are three kinds of general, exact methods to perform inference on a probabilistic graphical model: the naive one, one called variable elimination, and the other one is a message-passing technique (presented in [WJ08, KF09] for example).

The naive inference algorithm consists in computing every joint probability, as we have done in Fig. 2. An inference task can be modelled as searching for $\Pr(\theta|D) = \frac{\Pr(\theta,D)}{\Pr(D)}$. Let $X \in \{\{\theta, D\}, \{D\}\}$. We show how to compute $\Pr(X = x)$. Let $\mathcal{X}$ be the set of random variables of the considered model, and $Y = \mathcal{X} \setminus X$. Then, by the sum rule, $\Pr(X = x) = \sum_y \Pr(X = x, Y = y)$. Now that we have computed both $\Pr(\theta, D)$ and $\Pr(D)$, we can just divide these two terms to get our result.

2.4 Approximate methods for inference

As we have seen, exact inference can be really costly, and approximate methods are an interesting alternative. We can say that there are two main classes of approximate methods: one called sampling, and the other one called variational inference.

2.4.1 Sampling methods

The idea of sampling-based methods is to draw many instances of our model to have an idea of the situation. The instances are sometimes called particles, and sampling methods are sometimes named particle-based methods. The scope of sampling methods is bigger than estimating the probability of an event, the general framework is about computing the expected value μ of a function $f(x)$ under some probability density $p(x)$, that is, $\mu = \mathbb{E}_{p(x)}[f(x)] = \int f(x)p(x)dx$. In particular, computing the probability $P(X = A)$ is equivalent to computing $\mathbb{E}[I_A(X)]$, where I is the usual indicator function.

The most basic sampling algorithm does the following: let $n \in \mathbb{N}$, we define $\hat{f}_n(x) = \frac{1}{n}\sum_{i=1}^{n} f(x_i)$ with $x = (x_1, \ldots, x_n)$, the x_i drawn according to the probability density p. Then, $\hat{f}_n(X)$, our *estimator* of μ, is unbiased: if X is a random vector of size n following the probability density p^n, then $\mathbb{E}_{X\sim p^n}[\hat{f}_n(X)] = \mu$. Thus, using the strong law of large numbers, we know that $\Pr(\ lim_{n\to+\infty}\hat{f}_n(X) = \mu) = 1$, so this approximation is good, at least asymptotically.

However, this method is not interesting to compute a posterior distribution $\Pr(X|A)$. One simple idea, called rejection sampling, is to sample our x_i's according to $\Pr(X)$, but then keep them only if they satisfy the event A. Although this is simple, it is quite inefficient if $\Pr(X|A)$ is much smaller than $\Pr(X)$.

In our example of a Bayes Network, sampling the x_i's according to $\Pr(X)$ is done by going through the graph and making choices at random following the conditional probabilities. For example, we would choose first if it is cloudy (with probability 0.4), then if the sprinkler is on and if it is raining (with probability 0.1 and 0.8 if the sky is cloudy), and so on. Doing the rejection part consists just in checking which vector of events we picked, and then keeping this vector only if it satisfies the conditional event.

More elaborate sampling methods include importance sampling, as well as Monte Carlo Markov Chain-based methods (MCMC).

2.4.2 Variational methods

The idea of variational methods for inference is to find the best element q of an approximation family $\mathcal{Q}$ to approximate a posterior distribution. To measure what is the "best" element of our approximation family, we will use a specific distance. However, the usual distances will still involve our posterior distribution, which is usually intractable – that is, not computable in a reasonable time. We will instead use a function called the relative entropy, or the KL-divergence, defined below. With some manipulations, we will not need to compute the posterior distribution. The approximation family should also be less complex than the posterior: otherwise, this method is not interesting, because it would be not efficient.

Definition 2 (KL-divergence)**.** Let p and q be two probability density functions over the same space. The Kullback-Leibler divergence is defined as:

$$\mathsf{KL}(p||q) = \int_{-\infty}^{+\infty} p(x) \log \frac{p(x)}{q(x)} dx$$

If P and Q are discrete probability distributions, we have:

$$\mathsf{KL}(p||q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

Remark 1. We notice that to define the KL-divergence of two density functions, we need the assumption $q(x) = 0 \Rightarrow p(x) = 0$. This is equivalent to $\text{supp}(p) \subseteq \text{supp}(q)$, where $\text{supp}\, f = \{x \mid f(x) > 0\}$ is the support of the density f.

Let us suppose that we want to compute $p(\theta|D)$. In variational inference, we try to solve the following optimisation problem: $\text{argmin}_{q \in \mathcal{Q}} \mathsf{KL}(q||p)$. We now show how to transform this problem into a maximisation problem only involving the joint distribution, which is known.

We define the evidence lower bound to transform our problem:

Definition 3 (Evidence lower-bound)**.** $\mathcal{L}_{vi}(q) := \log p(D) - \mathsf{KL}(q(\theta)||p(\theta|D))$

Proposition 1.

$$\mathcal{L}_{vi}(q) = \int q(\theta) \log \frac{p(\theta, D)}{q(\theta)} d\theta$$

Proof.

$$\begin{aligned}
\log p(D) - \mathsf{KL}(q||p) &= \log p(D) \cdot \int q(\theta) d\theta - \int q(\theta) \log \frac{q(\theta)}{p(\theta|D)} d\theta \\
&= \int q(\theta) \log p(D) d\theta + \int q(\theta) \log \frac{p(\theta|D)}{q(\theta)} d\theta \\
&= \int q(\theta) \log \frac{p(\theta, D)}{q(\theta)} d\theta
\end{aligned}$$

□

Using Prop. 1, we see that $\underset{q \in \mathcal{Q}}{\text{argmin}}\, \mathsf{KL}(p||q) = \underset{q \in \mathcal{Q}}{\text{argmax}} \int q(\theta) \log \frac{p(\theta, D)}{q(\theta)} d\theta$. The advantage of the last formulation is that only the joint distribution of p appears, which is easier to compute than

the posterior. In our Bayes Network of Fig. 1, we have seen that sampling for the joint distribution just requires to make choices while going through the graph. On the contrary, sampling from the posterior distribution requires to perform rejection sampling, which is computationally expensive.

Now, to optimise $\mathcal{L}_{vi}(q)$, several techniques are possible, among which gradient ascent, as well as Lagrange multipliers.

3 A short introduction to probabilistic programming

As we have seen before, the algorithms to perform inference do not depend on the model. As long as our model is still a Bayes Network, we can change our model of Fig. 1, and apply the same algorithm to compute a posterior. We could for example add an event to take into account dew in the early morning. Thus, probabilistic graphical models separate our models from the algorithms we use to reason on these models. However, there are still different classes of models, for which different inference algorithms are necessary. This slows down the process of developing new models, and requires people with skills in inference algorithms to develop new models. The goal of probabilistic programming is to simplify this process by letting users encode their model, and leaving the inference to the computer. Probabilistic programs are more expressive than probabilistic graphical models, and are usually based on a functional programming language. In the following, we introduce the basics of a probabilistic programming language called Anglican, before explaining a way to perform inference on probabilistic programs.

3.1 Adding programming languages to probabilistic models

In this part, we present a probabilistic programming language called Anglican [WvdMM14], developed by researchers from the University of Oxford. Anglican is based on Clojure, a functional programming language. It handles continuous and discrete random variables, called distributions, and proposes different inference techniques. Anglican extends the syntax of Clojure using some new keywords:

- `(defquery myquery [args] <body>)` defines a probabilistic query, having some arguments `args`. We can then call different inference algorithms on this query using `doquery`,
- `(sample <dist>)` samples an element following the provided distribution,
- `(observe <dist> <value>)` conditions the query by the event "`dist` = `value`". We will see this in more detail in the next section,
- `(predict <expr>)` is the return parameter of the query. It asks for the distribution probability of the expression `<expr>`.

We now explain in detail the program encoding a Bayes Network presented in Listing. 1.

We start by defining a query inside Anglican, called `bayes-net`, and having two arguments, `sprinkler` and `wet-grass`. We then define four variables: two scalars and two probability distributions, which are equivalent to the tables given in the Bayes Net definition. The construction `(cond b1 c1 b2 c2)` is the usual `(if b1 then c1 else if b2 then c2)`. In lines 21 and 22, we force the query to observe the state of the sprinkler and the grass according to the arguments of the query. At the end of the query, we ask for the prediction of the event `is-raining`. We call the inference algorithm called "pimh" (meaning "Particle Independent Metropolis-Hastings", which is a sampling technique) using the `doquery` keyword, to sample elements from this query. The result of one execution of the code in Listing. 1 is: `:predicts [[:is-raining true]], :log-weight 0.0`

`2878, :predicts [[:is-raining false]], :log-weight 0.0 7122`. Indeed, 28.78% is quite a good approximation of $\Pr(R = T|W = T, S = T)$, which was 28.98%: the relative error is 0.69%. On 10 executions, we got the following results: 28.78%, 28.68%, 28.78%, 28.78%, 29.04%, 29.34%, 29.29%, 28.74%, 28.63%, 29.75%, having mean 28.981%.

3.2 Inference on probabilistic programs

The basic idea of inference in probabilistic programs is to collect different traces sampled from the posterior distribution.

Using the idea of rejection sampling, inference on probabilistic programs is just collecting valid traces of a probabilistic program. We describe a general procedure sampling a valid trace from a probabilistic program in Algorithm. 1.

Algorithm 1: An algorithm sampling from a program

Input: sequence of command $c = (c_1, \ldots, c_n)$
Output: a valid trace for the program c, satisfying the given observations

```
i = 1
while i < n do
    if c_i matches sample dist then
        just sample from dist
    else if c_i matches observe dist value then
        sample a value v from dist
        if v ≠ value then
            Reject the trace
            Restart the procedure
    else if c_i matches predict expr then
        store the result of the predict statement as the evaluation of expr
    else
        just interpret the program as usual
    i = i + 1
end
```

As we have seen before, rejection sampling can be quite inefficient, but it is easy to describe.

4 Variational Inference for Probabilistic Programs

Now that we have seen the basics of probabilistic programming, we present the theoretical results we established during my internship. We first encode probabilistic programs into a probabilistic transition system (PTS). We define what an approximation of our PTS is, before expressing the variational inference problem we want to solve. We then present new expressions of the Evidence Lower Bound (ELBO), before giving a new inference algorithm derived from the theoretical expression of the ELBO.

4.1 The PTS framework

In this section, we formalise a probabilistic program as a probabilistic transition system. We use a measure-theoretic approach. Some reminders of measure theory are presented in appendix A. We

first define the notion of transition density. It is just a transition function which is probabilistic and described as a density.

Definition 4. A *transition density* on a measure space $(S, \mathcal{F}, \mu)$ is a measurable function $k : S \times S \to \mathbb{R}_+$ such that:

1. for all $s \in S$,
$$\int_S k(s, s')\, \mu(ds') = 1;$$
2. for all measurable subset $S_0 \subseteq S$ and $r \in \mathbb{R}_+$, the following set is measurable:
$$\left\{ s \;\middle|\; \int_{S_0} k(s, s')\, \mu(ds') < r \right\}$$

Using this notion of transition, we can now define the notion of probabilistic transition system:

Definition 5. A *probabilistic transition system* (in short, PTS) is a septuple $M = (S, \mathcal{F}, \mu, f, \delta, \psi, F)$ which consists of the following data:

- $(S, \mathcal{F}, \mu)$ is a measure space such that μ is σ-finite. S is the set of states, and an element of S intuitively represents the values of both the program variables and the program counter.
- f corresponds to a probability density on the initial states: f is a non-negative measurable function from S to $\mathbb{R}_+$ such that
$$\int_S f(s)\, \mu(ds) = 1.$$
- δ is a transition density on $(S, \mathcal{F}, \mu)$.
- ψ is a measurable function from S to $\mathbb{R}_+$. It determines the scores of the states. Intuitively, δ will be used to model the `sample` statements, and ψ will be used to model the `observe` statements. We will see this in Example 1.
- $F \subseteq S$ is a measurable subset of S and denotes the set of final states.

Assume that we are given a PTS: $M = (S, \mathcal{F}, \mu, \delta, \psi, F)$ We use the following measure space for the nonempty finite execution traces of M:

$$\tau \in T = \bigsqcup_{1 \leq n < \infty} S^n$$

The σ-algebra $\mathcal{G}$ and the measure ν of T are obtained by the standard constructions for countable disjoint sums and finite products of σ-finite measure spaces.

Using the PTS M, we can define two measurable functions. Let $n \geq 0$ and $\tau = s_0 \dots s_n \in T$:

$$\Delta(\tau) = \prod_{0 \leq i < n} [s_i \notin F] \cdot [s_n \in F] \cdot f(s_0) \cdot \prod_{0 \leq i < n} \delta(s_i, s_{i+1}) \qquad \Psi(\tau) = \prod_{0 \leq i \leq n} \psi(s_i)$$

Δ can be seen as the density of the prior probability on the execution traces of M. Here $[s_i \notin F]$ is a notation for the indicator function taking value 1 if $s_i \notin F$, and 0 if $s_i \in F$. Δ forces the execution trace to be a valid one: it should start in an initial state, and it finishes in the first final state it encounters. Ψ can be seen as the density of the likelihood term.

We also defined a notion of well-formedness of our PTS M, in order to define a posterior density probability.

Definition 6. A PTS M is *well-formed* if its Δ and Ψ on the associated measure space $(T, \mathcal{G}, \nu)$ of finite traces satisfy the following condition:

$$0 < \int_T \Big(\Delta(\tau) \cdot \Psi(\tau)\Big)\, \nu(d\tau) < \infty.$$

For such a well-formed PTS M, we define a measurable function $\Pi : T \to \mathbb{R}_+$ as follows:

$$Z = \int_T \Big(\Delta(\tau) \cdot \Psi(\tau)\Big)\, \nu(d\tau), \qquad \Pi(\tau) = \frac{\Delta(\tau) \cdot \Psi(\tau)}{Z},$$

and call Π the *posterior density*.

Example 1. We present a really simple example of a conversion of a probabilistic program into a PTS. We consider the following program (where the ^{l_i} are program labels):

Listing 2: A fairness issue

```
(defquery coin []
  (let [l1is-fair (sample (flip 0.9))
        l2coin (if is-fair
                 (flip 0.5)
                 (flip 0.95))]
    l3(observe coin 1)
    l4(observe coin 1)
    l5(predict is-fair))l6)
```

This program describes that we have a coin, which has a 90% chances of being fair, and a 10% chance of being biased (then it follows a Bernoulli law of parameter 0.95), and that when we toss a coin two times, it gives two heads. We search for the probability that this coin is fair or not.

Here, our set of states is $S = [\![1, 6]\!] \times \{0, 1\}$, where $[\![1, 6]\!]$ corresponds to the program labels, and $\{0, 1\}$ corresponds to the value of the variable `is-fair`, 0 meaning false. We define $f((1, 0)) = 1$ (so $(1, 0)$ is the only initial state), and $F = \{(6, 0), (6, 1)\}$. Let $\tau_1 = ((1, 0), (2, 0), (3, 0), (4, 0), (5, 0), (6, 0))$ and $\tau_2 = ((1, 0), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1))$, we have $\Delta(\tau_1) = \delta((1, 0), (2, 0)) = 0.1$, and $\Delta(\tau_2) = \delta((1, 0), (2, 1)) = 0.9$ (we just choose `is-fair`). On the other hand, $\Psi(\tau_1) = \psi((4, 0))\psi((5, 0)) = 0.95^2$, and $\Psi(\tau_2) = \psi((4, 1))\psi((5, 1)) = 0.5^2$ (because we observe two times a "head" event for the coin). Thus, $Z = \Delta(\tau_1)\Psi(\tau_1) + \Delta(\tau_2)\Psi(\tau_2) = 0.31525$, so $\Pi(\tau_1) = 28.62\%$, and $\Pi(\tau_2) = 71.37\%$, and therefore coin is probably still fair. Let us do a quick check: if we had n `(observe coin 1)`, we would have $Z = 0.1 \cdot 0.95^n + 0.9 \cdot 0.5^n$ and:

$$\Pi(\tau_1) = \frac{1}{1 + 9 \cdot (\frac{10}{19})^n} \qquad \Pi(\tau_2) = \frac{1}{1 + \frac{1.9^n}{9}}$$

In that case, if $n = 11$, we already have a 99.23% chance that the coin is unfair.

4.2 Approximating the PTS

In variational inference, the goal is to minimise the KL divergence over an approximation family. Here, we define the approximation family for our PTS. We want to approximate a well-formed PTS

M. To do this, we suppose that we have a family consisting of pairs of densities and transition densities parametrized by θ: $\{(f_\theta, \delta_\theta)\}_{\theta\in\Theta}$ on the measure space $(S, \mathcal{F}, \mu)$. Each pair induces another PTS $M_\theta = (S, \mathcal{F}, \mu, f_\theta, \delta_\theta, \psi_\theta, F)$, with $\psi_\theta(s) = 1, \forall s \in S$. Then, we have:

$$\Delta_\theta(\tau) = \prod_{0\le i<n} [s_i \notin F] \cdot [s_n \in F] \cdot f_\theta(s_0) \cdot \prod_{0\le i<n} \delta_\theta(s_i, s_{i+1}), \qquad \Psi_\theta(\tau) = \prod_{0\le i\le n} \psi_\theta(s_i) = 1.$$

Assuming that $\Pi_\theta(\tau) = \Delta_\theta(\tau)$, we will need to compute $\mathsf{KL}(\Delta_\theta||\Pi)$. Using Remark 1, we need to be sure that $\mathrm{supp}(\Delta_\theta) \subseteq \mathrm{supp}(\Pi)$. Thus, Δ_θ underapproximates Π. We will also use gradient ascent, so we will need our approximation family to be differentiable. To enforce all these conditions, we define the well-formedness of our approximation family:

Definition 7. $\{(f_\theta, \delta_\theta)\}_{\theta\in\Theta}$ is *a well-formed approximation family* if

1. for all $\theta \in \Theta$,
$$1 = \int_T \Delta_\theta(\tau)\, \nu(d\tau),$$
2. $f(s) \cdot \psi(s) = 0 \implies f_\theta(s) = 0$ for all $s \in S$ and $\theta \in \Theta$;,
3. $\delta(s, s') \cdot \psi(s') = 0 \implies \delta_\theta(s, s') = 0$ for all $s, s' \in S$ and $\theta \in \Theta$;,
4. Θ is an open subset of $\mathbb{R}^n$ for some n and contains $\mathbf{0}$,
5. for all $s, s' \in S$, the functions $\theta \longmapsto f_\theta(s)$ and $\theta \longmapsto \delta_\theta(s, s')$ from Θ to $\mathbb{R}_+$ are differentiable.

In particular, this ensures that the KL-divergence is well-defined:

Lemma 2. *For all $\theta \in \Theta$,* $\mathrm{supp}\,\Delta_\theta \subseteq \mathrm{supp}\,\Pi$. *That is*

$$\forall \tau \in T, \Pi(\tau) = 0 \implies \Delta_\theta(\tau) = 0.$$

Proof. Consider $\theta \in \Theta$ and $\tau \in T$ such that $\Pi(\tau) = 0$. Let $s_0 \dots s_n = \tau$. Since $\Pi(\tau) = 0$:

$$\prod_{0\le i<n} [s_i \notin F] \cdot [s_n \in F] \cdot f(s_0) \cdot \prod_{0\le i<n} \delta(s_i, s_{i+1}) \cdot \prod_{0\le i\le n} \psi(s_i) = 0.$$

This means that at least one of the following terms is 0:

$$\prod_{0\le i<n} [s_i \notin F], \qquad [s_n \in F], \qquad f(s_0) \cdot \psi(s_0), \qquad \prod_{0\le i<n} (\delta(s_i, s_{i+1}) \cdot \psi(s_{i+1})).$$

Thus,

$$\Delta_\theta(\tau) = \prod_{0\le i<n} [s_i \notin F] \cdot [s_n \in F] \cdot f_\theta(s_0) \cdot \prod_{0\le i<n} \delta_\theta(s_i, s_{i+1}) = 0.$$

□

4.3 Expressing the ELBO in the PTS

Now that everything is well-defined, we can formulate our inference problem as searching for $\operatorname*{argmin}_{\theta\in\Theta} \mathsf{KL}(\Delta_\theta||\Pi) = \operatorname*{argmax}_{\theta\in\Theta} \mathcal{L}_{vi}(\Delta_\theta)$. Here, the evidence lower bound is:

$$\mathcal{L}_{vi}(\Delta_\theta) = \int_T \Delta_\theta(\tau) \log\left(\frac{\Delta(\tau)\cdot\Psi(\tau)}{\Delta_\theta(\tau)}\right)\nu(d\tau) = \mathbb{E}_{\Delta_\theta}\left[\log\frac{\Delta(\tau)\cdot\Psi(\tau)}{\Delta_\theta(\tau)}\right]$$

Using some tricks, we can also simplify the expression of $\nabla_\theta\mathcal{L}_{vi}$:

Proposition 3. *For all $\theta\in\Theta$, we have that*

$$\nabla_\theta\mathcal{L}_{vi} = \mathbb{E}_{\Delta_\theta}\left[\nabla_\theta\left[\log\Delta_\theta(\tau)\right]\cdot\log\left(\frac{\Delta(\tau)\cdot\Psi(\tau)}{\Delta_\theta(\tau)}\right)\right].$$

Proof.

$$\begin{aligned}
\nabla_\theta\mathcal{L}_{vi} &= \nabla_\theta\left[\int_T \Delta_\theta(\tau)\cdot\log\left(\frac{\Delta(\tau)\cdot\Psi(\tau)}{\Delta_\theta(\tau)}\right)\nu(d\tau)\right]\\
&= \int_T \nabla_\theta\left[\Delta_\theta(\tau)\cdot\log\left(\frac{\Delta(\tau)\cdot\Psi(\tau)}{\Delta_\theta(\tau)}\right)\right]\nu(d\tau)\\
&= \int_T \Delta_\theta(\tau)\cdot\nabla_\theta\left[\log\left(\frac{\Delta(\tau)\cdot\Psi(\tau)}{\Delta_\theta(\tau)}\right)\right]\nu(d\tau) + \int_T \nabla_\theta\left[\Delta_\theta(\tau)\right]\cdot\log\left(\frac{\Delta(\tau)\cdot\Psi(\tau)}{\Delta_\theta(\tau)}\right)\nu(d\tau)\\
&= -\int_T \Delta_\theta(\tau)\cdot\nabla_\theta\left[\log\Delta_\theta(\tau)\right]\nu(d\tau) + \int_T \Delta_\theta(\tau)\cdot\nabla_\theta\left[\log\Delta_\theta(\tau)\right]\cdot\log\left(\frac{\Delta(\tau)\cdot\Psi(\tau)}{\Delta_\theta(\tau)}\right)\nu(d\tau)\\
&= -\int_T \Delta_\theta(\tau)\cdot\frac{\nabla_\theta\cdot\Delta_\theta(\tau)}{\Delta_\theta(\tau)}\nu(d\tau) + \mathbb{E}_{\Delta_\theta}\left[\nabla_\theta\left[\log\Delta_\theta(\tau)\right]\cdot\log\left(\frac{\Delta(\tau)\cdot\Psi(\tau)}{\Delta_\theta(\tau)}\right)\right]\\
&= \underbrace{-\nabla_\theta\underbrace{\int_T \Delta_\theta(\tau)\,\nu(d\tau)}_{1}}_{0} + \mathbb{E}_{\Delta_\theta}\left[\nabla_\theta\left[\log\Delta_\theta(\tau)\right]\cdot\log\left(\frac{\Delta(\tau)\cdot\Psi(\tau)}{\Delta_\theta(\tau)}\right)\right]
\end{aligned}$$

□

We now use the specificity of our PTS models to find other expressions of $\mathcal{L}_{vi}$. We define two families of functions, $\{a^i_\theta : S\to\mathbb{R}_+\}_{i\in\mathbb{N}}$ and $\{b^i_\theta : S\to\mathbb{R}_+\}_{i\in\mathbb{N}}$. $a^i_\theta(s)$ expresses the probability that the i-th state of the execution is defined and is s. $b^i_\theta(s)$ represents the probability that the execution starting in s terminates after i steps. The goal of this definition is to be able to split the probability to get a trace $\tau = s_1\dots s_n$ into an expression using $a^k_\theta(s_k), \delta_\theta(s_k, s_{k+1}), b^{n-k}_\theta(s_{k+1})$.

These two classes of functions are defined inductively as follows:

$$a^0_\theta(s') = f_\theta(s')\cdot[s'\notin F] \qquad a^{i+1}_\theta(s') = \int a^i_\theta(s)\cdot\delta_\theta(s,s')\cdot[s'\notin F]\,\mu(ds)$$

$$b^0_\theta(s) = [s\in F] \qquad b^{i+1}_\theta(s) = \int [s\notin F]\cdot\delta_\theta(s,s')\cdot b^i_\theta(s')\,\mu(ds')$$

Using this inductive definition, we can easily deduce the following expanded expressions:

Lemma 4. *For all $n \geq 1$,*

$$a_\theta^n(s) = \int \Big(f_\theta(s_0) \cdot [s_0 \notin F] \cdot \prod_{i=0}^{n-2} (\delta_\theta(s_i, s_{i+1}) \cdot [s_{i+1} \notin F]) \cdot \delta_\theta(s_{n-1}, s) \cdot [s \notin F] \Big) \, \mu^n(d(\mathbf{s}_{0:(n-1)}))$$

$$b_\theta^n(s) = \int \Big([s \notin F] \cdot \delta_\theta(s, s_1) \cdot \prod_{i=1}^{n-1} ([s_i \notin F] \cdot \delta_\theta(s_i, s_{i+1})) \cdot [s_n \in F] \Big) \, \mu^n(d(\mathbf{s}_{1:n}))$$

In the following, we also use the following four functions:

$$A_\theta(s) = \sum_{i=0}^{\infty} a_\theta^i(s) \qquad B_\theta(s) = \sum_{i=0}^{\infty} b_\theta^i(s)$$

$$g(s) = f(s) \cdot \psi(s) \qquad \kappa(s, s') = \delta(s, s') \cdot \psi(s')$$

We express parts of the integrand of $\mathcal{L}_{vi}$, using T_n for $n \geq 0$:

$$T_n(\tau) = [|\tau| = n + 1] \cdot \Delta_\theta(\tau) \cdot \log \left(\frac{\Delta(\tau) \cdot \Psi(\tau)}{\Delta_\theta(\tau)} \right)$$

In particular, we get the following result:

Lemma 5. *For all $n \geq 0$,*

$$\int T_n(\tau) \, \nu(d\tau) = \int f_\theta(s) \log \frac{g(s)}{f_\theta(s)} b_\theta^n(s) \, \mu(ds) + \sum_{k=0}^{n-1} \int \delta_\theta(s, s') \log \frac{\kappa(s, s')}{\delta_\theta(s, s')} a_\theta^k(s) b_\theta^{n-k-1}(s') \, \mu^2(d(s, s'))$$

Proof. The proof is presented in Sec. B.1. □

Now, to express $\mathcal{L}_{vi}$ with the family of $(T_n)_{n \in \mathbb{N}}$, we need the following assumptions to hold:

Assumption 1.

$$n \cdot \int a_\theta^i(s) \mu(ds) \to 0 \text{ as } n \to \infty \tag{2}$$

$$0 < \int A_\theta(s) \mu(ds) < +\infty \tag{3}$$

$$\sum_{i=1}^{\infty} \int T_n(\tau) \, \nu(d\tau) = \int \sum_{i=1}^{\infty} T_n(\tau) \, \nu(d\tau) \tag{4}$$

$$\int f_\theta(s) \cdot \log \frac{g(s)}{f_\theta(s)} \, \mu(ds) < \infty \tag{5}$$

$$\int \delta_\theta(s, s') \cdot \log \frac{\kappa(s, s')}{\delta_\theta(s, s')} \cdot A_\theta(s) \, \mu^2(d(s, s')) < \infty \tag{6}$$

Theorem 6 (A first expression of $\mathcal{L}_{vi}$)**.** *Under the assumptions above, we can show that:*

$$\mathcal{L}_{vi} = \int f_\theta(s) \cdot \log \frac{g(s)}{f_\theta(s)} \, \mu(ds) + \int \delta_\theta(s, s') \cdot \log \frac{\kappa(s, s')}{\delta_\theta(s, s')} \cdot A_\theta(s) \, \mu^2(d(s, s')).$$

Proof. The proof is quite long and difficult, it is omitted from this document. It is easy to show that (this is shown in Appendix B.2):

$$\mathcal{L}_{vi} = \int f_\theta(s) \cdot \log \frac{g(s)}{f_\theta(s)} \cdot B_\theta(s)\, \mu(ds) + \int \delta_\theta(s, s') \cdot \log \frac{\kappa(s, s')}{\delta_\theta(s, s')} \cdot A_\theta(s) \cdot B_\theta(s')\, \mu^2(d(s, s'))$$

Then, we initially thought that $B_\theta(s) = 1$, but we were not able to prove this. We had to prove a weaker, and more difficult result. We needed all the assumptions presented above to make the proof. □

After deriving this result, we tried to find simpler expression of the Evidence Lower Bound $\mathcal{L}_{vi}$, under some assumptions. We tried to see what happens:

- when $\delta_\theta(s, s') = f_\theta(s')$. That is, the transition density does not depend on the pre-state s,
- when the transition density $\delta_\theta(s, s')$ depends on s' and on the program label of s,
- when at most one variable changes at each transition.

We obtained various results, but the computations where long and not really useful. We searched for another general, simplified expression of $\mathcal{L}_{vi}$. We tried to replace s by an s_i, and then force $s = s_i$. To do this, we introduced the Dirac distribution translated by s, written $\delta_s(\cdot)$. This distribution has the following property: $\int f(s_i)\delta_s(ds_i) = f(s)$.

We first introduce two new quantities d_θ and e_θ to express the partial sums of A_θ. Then we derive a recurrence relation between a_θ, d_θ and e_θ before proving the new expression of A_θ.

Definition 8.

$$d^n_\theta(s) = \sum_{k=0}^{n} \int f_\theta(s_0) \prod_{i=0}^{n-1} (\delta_\theta(s_i, s_{i+1})[s_i \notin F])[s_n \notin F] \mu^k(ds_{0:(k-1)}) \delta_s(ds_k) \mu^{n-k}(ds_{k+1:n})$$

$$e^n_\theta(s) = \sum_{k=0}^{n} \sum_{i=0}^{k-1} \int f_\theta(s_0) \prod_{j=0}^{k-1} (\delta_\theta(s_j, s_{j+1})[s_j \notin F])[s_k \in F] \mu^i(ds_{0:i-1}) \delta_s(ds_i) \mu^{k-i}(ds_{i+1:k})$$

Proposition 7 (Recurrence relation)**.**

$$\forall n \in \mathbb{N}, d^{n+1}_\theta(s) + e^{n+1}_\theta(s) = a^{n+1}_\theta(s) + d^n_\theta(s) + e^n_\theta(s)$$

Proof. This is a simple proof, shown in Appendix. B.3. □

Proposition 8.

$$\sum_{k=0}^{n} a^k_\theta(s) = d^n_\theta(s) + e^n_\theta(s)$$

Proof.

$$\begin{aligned} d^n_\theta(s) + e^n_\theta(s) &= \sum_{k=1}^{n} a^k_\theta(s) + d^0_\theta(s) + e^0_\theta(s) \\ &= \sum_{k=1}^{n} a^k_\theta(s) + \underbrace{\int f_\theta(s_0)[s_0 \notin F]\delta_s(ds_0)}_{a^0_\theta(s)} \end{aligned}$$

□

Corollary 9. *Let*

$$D_\theta(s) = \lim_{n\to+\infty} d_\theta^n(s) \qquad\qquad E_\theta(s) = \lim_{n\to+\infty} e_\theta^n(s)$$

D_θ and E_θ are well-defined, and $A_\theta(s) = D_\theta(s) + E_\theta(s)$

We give further simplifications for the expressions of D_θ and E_θ:

Proposition 10.

$$D_\theta = 0$$

Proof. By the assumption given in Eq.(3), $\int A_\theta(s)\mu(ds) < +\infty$, so: $\lim_{k\to+\infty} \int a_\theta^k(s)\mu(ds) = 0$. We also know that: $0 \leq d_\theta^n(s) \leq \int a_\theta^n(s)\mu(ds)$, so $\forall s \in S$, $D_\theta(s) = 0$. □

Proposition 11. *Let*

$$h_\theta(s) = \int \delta_\theta(s, s') \log \frac{\kappa(s, s')}{\delta_\theta(s, s')} \mu(ds').$$

We have:

$$\int E_\theta(s) h_\theta(s) \mu(ds) = \int \Delta_\theta(\tau) \Big(\sum_{i=1}^{\tau-1} h_\theta(\tau_i) \Big) \nu(d\tau)$$

Proof. We start with a simple observation on the use of a Dirac distribution:

$$\iint \alpha(s_i)\beta(s)\delta_s(ds_i)\mu(ds) = \int \alpha(s)\beta(s)\mu(ds) = \int \alpha(s_i)\beta(s_i)\mu(ds_i)$$

This is exactly what we use in the following:

$$\begin{aligned}
&\int E_\theta(s) h_\theta(s) \mu(ds) \\
&= \int \sum_{k=0}^{+\infty} \sum_{i=0}^{k-1} \left(h_\theta(s) f_\theta(s_0) \prod_{j=0}^{k-1} (\delta_\theta(s_j, s_{j+1})[s_j \notin F]) [s_k \in F] \mu^i(ds_{0:i-1}) \mu^{k-i}(ds_{i+1:k}) \delta_s(ds_i) \mu(ds) \right) \\
&= \int \sum_{k=0}^{+\infty} \left(\sum_{i=0}^{k-1} h_\theta(s_i) \right) \Delta_\theta(s_0 \cdots s_k) \mu^{k+1}(ds_{0:k}) \\
&= \int \Delta_\theta(\tau) \Big(\sum_{i=1}^{|\tau|-1} h_\theta(\tau_i) \Big) \nu(d\tau)
\end{aligned}$$

□

Theorem 12 (Second expression of the ELBO)**.** *Now, the ELBO has the following expression:*

$$\mathcal{L}_{vi} = \int f_\theta(s) \log \frac{g(s)}{f_\theta(s)} \mu(ds) + \int \Delta_\theta(\tau) \left(\sum_{i=1}^{|\tau|-1} h_\theta(\tau_i) \right) \nu(d\tau),$$

Here:

$$h_\theta(s) = \int \delta_\theta(s, s') \log \frac{\kappa(s, s')}{\delta_\theta(s, s')} \mu(ds')$$

Theorem 13. *The gradient of the ELBO is:*

$$\nabla_\theta \mathcal{L}_{vi} = \mathbb{E}_{f_\theta(s)}\left[\nabla_\theta \log f_\theta(s) \log \frac{f_\theta(s)}{g(s)}\right] + \mathbb{E}_{\Delta_\theta(\tau)}\left[\nabla_\theta \log \Delta_\theta(\tau) \left(\sum_{i=1}^{|\tau|-1} h_\theta(\tau_i)\right)\right] + \mathbb{E}_{\Delta_\theta(\tau)}\left[\sum_{i=1}^{|\tau|-1} \nabla_\theta h_\theta(\tau_i)\right]$$

With:

$$\nabla_\theta h_\theta(\tau_i) = \mathbb{E}_{\delta_\theta(\tau_i, s')}\left[\nabla_\theta \log \delta_\theta(\tau_i, s') \log \frac{\kappa(\tau_i, s')}{\delta_\theta(\tau_i, s')}\right]$$

Remark 2. We prefer to write expressions with expectations, since we can estimate the quantities easily with sampling-based methods, as described in Sec. 2.4.1.

4.4 Derived algorithms

Using the expressions obtained in Theorems 12 and 13, we can now derive a more precise inference algorithm. The basic algorithm is the following:

Algorithm 2: Simple Stochastic Gradient Ascent

Input: Number N of steps
begin
  Choose $\theta \in \Theta$ randomly
  for $i = 1$ *to* N **do**
    $\theta := \theta + \eta(i) \cdot \nabla_\theta \mathcal{L}_{vi}(\Delta_\theta)$
  Return θ
end

Here, $\eta(i)$ is called the step size or the learning rate. According to [RGB14], Algorithm 2 converges to a maximum of $\mathcal{L}_{vi}$ when the learning rate satisfies the Robbins-Monro conditions:

$$\sum_{i=1}^{\infty} \eta(i) = \infty \qquad \sum_{i=1}^{\infty} \eta(i)^2 < \infty$$

Now we have formalised variational inference on probabilistic programs into a PTS, and derived expressions of both the ELBO and its gradient, we need to show how to compute these quantities so that the Stochastic Gradient Ascent algorithm can be useful. This can be done either by implementing the explicit expression of the gradient of $\mathcal{L}_{vi}$ provided in Theorem 13, or by implementing the computation of $\mathcal{L}_{vi}$ using Theorem 12, and letting a program compute the gradient using *automatic differentiation*.

4.4.1 Explicit computation of $\nabla_\theta \mathcal{L}_{vi}$

We can implement a procedure to compute $\nabla_\theta \mathcal{L}_{vi}$. In fact, what is usually done is an approximate computation of the gradient using *sampling*-based estimation. In other words, we use the following approximation (with $s_1, \ldots, s_n \in S^n$, sampled from f_θ):

$$\mathbb{E}_{f_\theta(s)}\left[\nabla_\theta \log f_\theta(s) \log \frac{f_\theta(s)}{g(s)}\right] \simeq \frac{1}{n} \sum_{i=1}^{n} \left(\nabla_\theta \log f_\theta(s_i) \log \frac{f_\theta(s_i)}{g(s_i)}\right)$$

We can also evaluate approximately the other terms of $\nabla_\theta \mathcal{L}_{vi}$ using this method.

4.4.2 Using automatic differentiation

Another method to compute the gradient is to use automatic differentiation. In short, Automatic differentiation [Ral81] uses the usual chain rule to compute the gradient of a procedure. Thus, we can implement a procedure computing $\mathcal{L}_{vi}(\Delta_\theta)$, and let an automatic differentiation library compute the gradient. We can also rewrite the ELBO using expectations to get:

$$\mathcal{L}_{vi} = \mathbb{E}_{f_\theta(s)}\left[\log \frac{g(s)}{f_\theta(s)}\right] + \mathbb{E}_{\Delta_\theta(\tau)}\left[\sum_{i=1}^{|\tau|-1} h_\theta(\tau_i)\right]$$

Estimating $h_\theta(s)$ using sampling is described in Algorithm 3, and estimating $\mathcal{L}_{vi}$ is presented in Algorithm 4. To sample a trace from Δ_θ, we just execute the approximated version of our probabilistic program (that is, we execute the PTS M_θ rather than M), using an interpreter similar to the one presented in Algorithm 1. We can use a Stochastic Gradient Ascent presented in Algorithm 2, where the gradient is computed by an automatic differentiation library called on the function computing $\mathcal{L}_{vi}$.

Algorithm 3: Computation of $h_\theta(s)$

Input: θ, s, N

h = 0
for *i = 1 to N* **do**
  Sample s' from $\delta_\theta(s, \cdot)$
  $h = h + \log \frac{\kappa(s,s')}{\delta_\theta(s,s')}$
return h / N

Algorithm 4: Computation of $\mathcal{L}_{vi}$

Input: θ, N_1, N_2

$l_1 = 0$
$l_2 = 0$
for *i = 1 to* N_1 **do**
  Sample a state s from f_θ
  $l_1 = l_1 + \log \frac{g(s)}{f_\theta(s)}$
for *i = 1 to* N_2 **do**
  Sample a trace τ from Δ_θ
  for *j = 1 to* $|\tau| - 1$ **do** $l_2 = l_2 + h_\theta(\tau_i)$
return $\frac{l_1}{N_1} + \frac{l_2}{N_2}$

5 Implementation and results

No implementation is ready yet due to a lack of time as well as some technical issues encountered at the end of my internship. My plan is to implement the algorithm presented above as another inference technique in Anglican. This way, I do not have to create a whole system with a parser and an interpreter. Still, I need to understand the structure of Anglican.

6 Related work

This part mentions various inference techniques for probabilistic programming, as well as various versions of black-box variational inference methods. I also read parts of general references on probabilistic graphical models [KF09] and variational inference [JGJS99, Bis06, WJ08].

In [Gha15], Ghahramani gives a short review of probabilistic programming, mentioning that probabilistic programming is a state-of-the-art advance in the field of probabilistic framework. Figaro [Pfe09] is a probabilistic programming language providing different sampling-based inference algorithms. Anglican [WvdMM14], Church [GMR+12] and Venture [MSP14] are all higher-order probabilistic programs, Venture being a successor of Church. Although they support a lot of different sampling algorithms, both Anglican and Church implement variational inference algorithms. We read papers about variational inference in probabilistic programs [WW13, KRGB15], but in our case, we tried a formal approach with longer derivations, that may be more precise than just optimising the basic ELBO. We hope that the simplifications we have made reduce the variance of the estimators of the ELBO.

We also studied papers on variational inference, but in the setting of probabilistic models rather than probabilistic programming. Usually, variational inference methods work only on a specific class of models. However, [RGB14] is about a global variational inference algorithm, not specific to a class of models, called "black-box variational inference". It also describes various techniques used to reduce the variance of the optimisation, because the gradient is estimated using samples. [HLR+16, LT16] present other black-box variational inference algorithms, where they use generalisations of the KL-divergence.

7 Conclusion

As we have seen, probabilistic programming simplifies the development of new probabilistic models by automating the inference. We hope that this approach will ease the development of new models, as it does not require knowledge in inference algorithms to create new models. During this internship, we expressed a probabilistic programming language into a probabilistic transition system (PTS). To use variational inference, we approximated the PTS by a simpler one, before comparing them using the evidence lower bound (ELBO). We simplified the expression of the ELBO in our setting to get something that is – hopefully – easier to compute and more precise, and derived a new inference algorithm from these expressions.

Future work I ran out of of time to implement this new inference algorithm into Anglican. This would be really interesting to implement, to get a real idea of the efficiency of this algorithm. It could also be interesting to generalise this algorithm to more general divergences, such as the Rényi divergence [LT16]. Another promising approach is to translate Cross-Entropy methods [dBKMR05] into our setting. Cross-Entropy is similar to Variational Inference, but the goal is to minimise $\mathsf{KL}(p||q)$ rather than $\mathsf{KL}(q||p)$. Thus, the approximating family should not underapproximate the support of the posterior, but overapproximate this. This might look insignificant, but at in program analysis overapproximations are more widespread and easier to compute than underapproximations.

References

[Bis06] Christopher M Bishop. *Pattern Recognition and Machine Learning*. Springer New York, 2006.

[Coo90] Gregory F. Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artif. Intell.*, 42(2-3):393–405, 1990.

[dBKMR05] Pieter-Tjerk de Boer, Dirk P. Kroese, Shie Mannor, and Reuven Y. Rubinstein. A tutorial on the cross-entropy method. *Annals OR*, 134(1):19–67, 2005.

[Gha15] Zoubin Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452–459, 2015.

[GMR+12] Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. Church: a language for generative models. *CoRR*, abs/1206.3255, 2012.

[GY07] Mark J. F. Gales and Steve J. Young. The application of hidden markov models in speech recognition. *Foundations and Trends in Signal Processing*, 1(3):195–304, 2007.

[HLR+16] José Miguel Hernández-Lobato, Yingzhen Li, Mark Rowland, Thang D. Bui, Daniel Hernández-Lobato, and Richard E. Turner. Black-box alpha divergence minimization. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 1511–1520, 2016.

[JGJS99] Michael I. Jordan, Zoubin Ghahramani, Tommi S. Jaakkola, and Lawrence K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, 1999.

[KF09] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models - Principles and Techniques*. MIT Press, 2009.

[KRGB15] Alp Kucukelbir, Rajesh Ranganath, Andrew Gelman, and David M. Blei. Automatic variational inference in stan. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 568–576, 2015.

[LT16] Yingzhen Li and Richard E. Turner. Rényi divergence variational inference. *CoRR*, abs/1602.02311, February 2016.

[MSP14] Vikash K. Mansinghka, Daniel Selsam, and Yura N. Perov. Venture: a higher-order probabilistic programming platform with programmable inference. *CoRR*, abs/1404.0099, 2014.

[Pfe09] Avi Pfeffer. Figaro: An object-oriented probabilistic programming language. *Charles River Analytics Technical Report*, 137, 2009.

[Ral81] Louis B. Rall. *Automatic Differentiation: Techniques and Applications*, volume 120 of *Lecture Notes in Computer Science*. Springer, 1981.

[RGB14] Rajesh Ranganath, Sean Gerrish, and David M. Blei. Black box variational inference. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, AISTATS 2014, Reykjavik, Iceland, April 22-25, 2014*, pages 814–822, 2014.

[Tao11] Terence Tao. *An introduction to measure theory*, volume 126. American Mathematical Soc., 2011.

[WJ08] Martin J. Wainwright and Michael I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008.

[WvdMM14] Frank Wood, Jan-Willem van de Meent, and Vikash Mansinghka. A new approach to probabilistic programming inference. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, AISTATS 2014, Reykjavik, Iceland, April 22-25, 2014*, pages 1024–1032, 2014.

[WW13] David Wingate and Theophane Weber. Automated variational inference in probabilistic programming. *CoRR*, abs/1301.1299, 2013.

A Some short reminders on measure theory and integration

In this appendix, we give some really short reminders on measure theory and integration, to fix notations used before. There are a lot of introductions to measure theory, such as [Tao11].

Definition 9 (σ-algebra)**.** Let X be a set. $\Sigma \subseteq \mathcal{P}(X)$ is a σ-algebra over X if:

- $\emptyset \in \Sigma$,
- $\forall B \in \Sigma$, $\overline{B} \in \Sigma$,
- if $\forall n \in \mathbb{N}, B_n \in \Sigma$, then $\bigcup_{n\in\mathbb{N}} B_n \in \Sigma$.

Note 1. In the following, X is a set, and Σ be a σ-algebra over X. (X, Σ) is called a measurable space.

Definition 10 (Measure)**.** $\mu : \Sigma \to \overline{\mathbb{R}}_+$ is called a measure if it satisfies the following properties:

- for all $E \in \Sigma$, $\mu(E) \geq 0$,
- $\mu(\emptyset) = 0$,
- if $(E_i)_{i\in I}$ is a pairwise disjoint countable collection, $\mu(\bigcup_{i\in I} E_i) = \sum_{i\in I} \mu(E_i)$.

Definition 11. A measure μ over (X, Σ) is *σ-finite* if $\mu(X) < +\infty$.

Definition 12 (Probability measure)**.** A probability measure μ is a measure such that $\mu(X) = 1$.

Example 2 (Dirac Measure)**.** Let $x \in X$, $A \in \Sigma$. We define the Dirac measure at point x, written δ_x , the following measure: $\delta_x(A) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A \end{cases}$

Definition 13 (Measure space)**.** A measure space is a triplet (X, Σ, μ), where X is a set, Σ is a σ-algebra over X and μ a measure over (X, Σ).

Definition 14 (Measurable set)**.** Let $A \subseteq X$, A is measurable if $A \in \Sigma$.

Definition 15 (Measurable function)**.** Let (Y, T) be a measurable space. $f : X \to Y$ is measurable if:

$$\forall E \in T, \{x \in X \mid f(x) \in E\} \in \Sigma$$

B Some missing proofs

B.1 A proof of Lemma 5

Proof. The following derivation shows the Lemma 5:

$$
\begin{aligned}
&\int T_n(\tau)\,\nu(d\tau) \\
&= \int \Big([|\tau| = n+1] \cdot \Delta_\theta(\tau) \cdot \log\Big(\frac{\Delta(\tau)\cdot\Psi(\tau)}{\Delta_\theta(\tau)}\Big)\Big)\,\nu(d\tau) \\
&= \int \prod_{i=0}^{n-1} [s_i \notin F] \cdot [s_n \in F] \cdot f_\theta(s_0) \cdot \prod_{j=0}^{n-1} \delta_\theta(s_j, s_{j+1}) \cdot \Big(\log\frac{g(s_0)}{f_\theta(s_0)} + \sum_{k=0}^{n-1} \log\frac{\kappa(s_k,s_{k+1})}{\delta_\theta(s_k,s_{k+1})}\Big)\,\mu^{n+1}(d(\mathbf{s}_{0:n})) \\
&= \int \prod_{i=0}^{n-1} [s_i \notin F] \cdot [s_n \in F] \cdot f_\theta(s_0) \cdot \prod_{j=0}^{n-1} \delta_\theta(s_j, s_{j+1}) \cdot \log\frac{g(s_0)}{f_\theta(s_0)}\,\mu^{n+1}(d(\mathbf{s}_{0:n})) \\
&\quad + \sum_{k=0}^{n-1} \int \prod_{i=0}^{n-1} [s_i \notin F] \cdot [s_n \in F] \cdot f_\theta(s_0) \cdot \prod_{j=0}^{n-1} \delta_\theta(s_j, s_{j+1}) \cdot \log\frac{\kappa(s_k,s_{k+1})}{\delta_\theta(s_k,s_{k+1})}\,\mu^{n+1}(d(\mathbf{s}_{0:n})) \\
&= \int f_\theta(s_0) \cdot \log\frac{g(s_0)}{f_\theta(s_0)} \cdot \prod_{i=0}^{n-1} \Big([s_i \notin F] \cdot \delta_\theta(s_i, s_{i+1})\Big) \cdot [s_n \in F]\,\mu^{n+1}(d(\mathbf{s}_{0:n})) \\
&\quad + \sum_{k=0}^{n-1} \int \delta_\theta(s_k,s_{k+1}) \cdot \log\frac{\kappa(s_k,s_{k+1})}{\delta_\theta(s_k,s_{k+1})} \cdot f_\theta(s_0) \cdot [s_0 \notin F] \cdot \prod_{i=0}^{k-1} \Big(\delta_\theta(s_i,s_{i+1}) \cdot [s_{i+1} \notin F]\Big) \\
&\qquad\qquad \cdot \prod_{j=k+1}^{n-1} \Big([s_j \notin F] \cdot \delta_\theta(s_j, s_{j+1})\Big) \cdot [s_n \in F]\,\mu^{n+1}(d(\mathbf{s}_{0:n})) \\
&= \int f_\theta(s_0) \cdot \log\frac{g(s_0)}{f_\theta(s_0)} \cdot b_\theta^n(s_0)\,\mu(ds_0) \\
&\quad + \sum_{k=0}^{n-1} \int \delta_\theta(s_k,s_{k+1}) \cdot \log\frac{\kappa(s_k,s_{k+1})}{\delta_\theta(s_k,s_{k+1})} \cdot a_\theta^k(s_k) \cdot b_\theta^{n-k-1}(s_{k+1})\,\mu^2(d(s_k,s_{k+1})) \\
&= \int f_\theta(s) \cdot \log\frac{g(s)}{f_\theta(s)} \cdot b_\theta^n(s)\,\mu(ds) + \sum_{k=0}^{n-1} \int \delta_\theta(s,s') \cdot \log\frac{\kappa(s,s')}{\delta_\theta(s,s')} \cdot a_\theta^k(s) \cdot b_\theta^{n-k-1}(s')\,\mu^2(d(s,s'))
\end{aligned}
$$

□

B.2 Easy expression of the ELBO

Theorem 14 ("Easy" expression of the ELBO)**.**

$$
\mathcal{L}_{vi} = \int f_\theta(s) \cdot \log\frac{g(s)}{f_\theta(s)} \cdot B_\theta(s)\,\mu(ds) + \int \delta_\theta(s,s') \cdot \log\frac{\kappa(s,s')}{\delta_\theta(s,s')} \cdot A_\theta(s) \cdot B_\theta(s')\,\mu^2(d(s,s'))
$$

Proof.

$$
\begin{aligned}
\mathcal{L}_{vi} &= \sum_{n=0}^{\infty} T_n \\
&= \sum_{n=0}^{\infty} \Big(\int f_\theta(s) \cdot \log \frac{g(s)}{f_\theta(s)} \cdot b_\theta^n(s)\, \mu(ds) \\
&\qquad + \sum_{k=0}^{n-1} \int \delta_\theta(s,s') \cdot \log \frac{\kappa(s,s')}{\delta_\theta(s,s')} \cdot a_\theta^k(s) \cdot b_\theta^{n-k-1}(s')\, \mu^2(d(s,s')) \Big) \\
&= \int f_\theta(s) \cdot \log \frac{g(s)}{f_\theta(s)} \cdot \Big(\sum_{n=0}^{\infty} b_\theta^n(s) \Big)\, \mu(ds) \\
&\quad + \sum_{n=0}^{\infty} \sum_{k=0}^{n-1} \int \delta_\theta(s,s') \cdot \log \frac{\kappa(s,s')}{\delta_\theta(s,s')} \cdot a_\theta^k(s) \cdot b_\theta^{n-k-1}(s')\, \mu^2(d(s,s')) \\
&= \int f_\theta(s) \cdot \log \frac{g(s)}{f_\theta(s)} \cdot \Big(\sum_{n=0}^{\infty} b_\theta^n(s) \Big)\, \mu(ds) \\
&\quad + \sum_{k=0}^{\infty} \sum_{n=k+1}^{\infty} \int \delta_\theta(s,s') \cdot \log \frac{\kappa(s,s')}{\delta_\theta(s,s')} \cdot a_\theta^k(s) \cdot b_\theta^{n-k-1}(s')\, \mu^2(d(s,s')) \\
&= \int f_\theta(s) \cdot \log \frac{g(s)}{f_\theta(s)} \cdot B_\theta(s)\, \mu(ds) \\
&\quad + \int \delta_\theta(s,s') \cdot \log \frac{\kappa(s,s')}{\delta_\theta(s,s')} \cdot \Big(\sum_{k=0}^{\infty} a_\theta^k(s) \cdot \Big(\sum_{n=k+1}^{\infty} b_\theta^{n-k-1}(s') \Big) \Big)\, \mu^2(d(s,s')) \\
&= \int f_\theta(s) \cdot \log \frac{g(s)}{f_\theta(s)} \cdot B_\theta(s)\, \mu(ds) + \int \delta_\theta(s,s') \cdot \log \frac{\kappa(s,s')}{\delta_\theta(s,s')} \cdot A_\theta(s) \cdot B_\theta(s')\, \mu^2(d(s,s'))
\end{aligned}
$$

□

B.3 Proof of Prop. 7

Proof.

$$
\begin{aligned}
&d_\theta^{n+1}(s) + e_\theta^{n+1}(s) \\
=&\sum_{k=0}^{n} \int f_\theta(s_0) \prod_{i=0}^{n} (\delta_\theta(s_i, s_{i+1})[s_i \notin F]) [s_{n+1} \notin F] \mu^k(ds_{0:k-1}) \delta_s(ds_k) \mu^{n-k}(ds_{k+1:n+1}) \\
&+ a_\theta^{n+1}(s) \\
&+ \sum_{k=0}^{n} \sum_{i=0}^{k-1} \int f_\theta(s_0) \prod_{j=0}^{k-1} (\delta_\theta(s_j, s_{j+1})[s_j \notin F]) [s_k \in F] \mu^i(ds_{0:i-1}) \delta_s(ds_i) \mu^{k-i}(ds_{i+1:k}) \\
&+ \sum_{i=0}^{n} \int f_\theta(s_0) \prod_{j=0}^{n} (\delta_\theta(s_j, s_{j+1})[s_j \notin F]) [s_{n+1} \in F] \mu^i(ds_{0:i-1}) \delta_s(ds_i) \mu^{n-i+1}(ds_{i+1:n+1}) \\
=&a_\theta^{n+1}(s) + e_\theta^{n}(s) \\
&+ \sum_{k=0}^{n} \int f_\theta(s_0) \prod_{i=0}^{n} (\delta_\theta(s_i, s_{i+1})[s_i \notin F]) ([s_{n+1} \notin F] + [s_{n+1} \in F]) \mu^k(ds_{0:k-1}) \delta_s(ds_k) \mu^{n-k+1}(ds_{k+1:n+1}) \\
=&a_\theta^{n+1}(s) + e_\theta^{n}(s) \\
&+ \sum_{k=0}^{n} \int f_\theta(s_0) \prod_{i=0}^{n-1} (\delta_\theta(s_i, s_{i+1})[s_i \notin F]) [s_n \notin F] \delta_\theta(s_n, s_{n+1}) \mu^k(ds_{0:k-1}) \delta_s(ds_k) \mu^{n-k+1}(ds_{k+1:n+1}) \\
=&a_\theta^{n+1}(s) + e_\theta^{n}(s) + \sum_{k=0}^{n} \int f_\theta(s_0) \prod_{i=0}^{n-1} (\delta_\theta(s_i, s_{i+1})[s_i \notin F]) [s_n \notin F] \mu^k(ds_{0:k-1}) \delta_s(ds_k) \mu^{n-k}(ds_{k+1:n}) \\
=&a_\theta^{n+1}(s) + e_\theta^{n}(s) + d_\theta^{n}(s)
\end{aligned}
$$

□