# The EfProb Library for Probabilistic Calculations*

**Kenta Cho and Bart Jacobs**

**Institute for Computing and Information Sciences**
**Radboud University, Nijmegen, The Netherlands**
{K.Cho,bart}@cs.ru.nl

**Abstract**

EfProb is an abbreviation of Effectus Probability. It is the name of a library for probability calculations in Python. EfProb offers a uniform language for discrete, continuous and quantum probability. For each of these three cases, the basic ingredients of the language are states, predicates, and channels. Probabilities are typically calculated as validities of predicates in states. States can be updated (conditioned) with predicates. Channels can be used for state transformation and for predicate transformation. This short paper gives an overview of the use of EfProb.

**Digital Object Identifier** 10.4230/LIPIcs.CALCO.2017.25

## 1 Introduction

The EfProb library provides an embbeded language in Python, for probability. The 'Ef' in EfProb stands for 'Effectus', and 'Prob' stand for 'Probability'. An effectus is an abstract (categorical) model that captures the essentials of discrete, continuous and quantum probability and logic [5, 1]. The EfProb library is based on this categorical model, providing a uniform approach to discrete, continuous, and quantum probability. However, in order to be able to use and understand the basic of the EfProb library it is not required to understand the underlying categorical semantics. The EfProb library makes it easy to model various problems in probability theory and to calculate and plot probability mass/density functions.

The aim of this short paper is to give a brief overview of EfProb, mainly by providing examples. The Python files that define EfProb are available online[1] together with an extensive manual that provides much more information. The core of EfProb has reached a reasonable level of stability, but development is still going on, driven both by practical and theoretical considerations.

We envision that the EfProb library could be useful in teaching the basics of probability theory from a unified perspective. At the same time the library could be useful in scientific research as well, in order to quickly model new examples and compute outcomes.

The library is not meant for large scale computation, for instance like those in data analytics. In our development of the library we prefer semantical clarity to execution speed. Indeed, the underlying channel-based semantics of EfProb is mathematically clear, in the sense that the library computes probabilities using the standard mathematical formulas, including, for instance, integration. This 'exact' computation is in contrast to approximate approaches via Monte Carlo sampling, which is used by languages such as BLOG [8], Church [3], and Anglican [11].

In practice, however, outcomes of EfProb are approximations too, since the library is based on numerical computation with floating-point numbers. Scaling is an issue in EfProb,

* The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement nr. 320571.
[1] At efprob.cs.ru.nl


exponential in the number of dimensions. In the continuous case, dimensions greater than 3 are already too slow to handle. It is important to keep in mind that the main functionality of the library is computing numerical outcomes, and not, for instance, logical reasoning.

The EfProb library is split into two parts, one for classical and one for quantum probability. The classical part integrates both discrete and continuous probability.

## 2 A uniform language

This section briefly describes the common language and notation that is used in EfProb for states, predicates, channels, *etc.* The description will be high-level, abstracting away from the different implementations for discrete, continuous and quantum probability.

### States

A state captures probabilities for elements of a certain domain. Classically, they are often called (probability) distributions. States in EfProb are closed under products, marginalisation, and convex sum. The product of two states `s`, `t` is written as `s @ t`. It involves the cartesian product of the underlying domains. Given a 'joint' state on a product domain, one can marginalise the state, for which a post-script notation is used. For instance, if `s` is a state on the product of three domains `D1`, `D2`, `D2`, then one can marginalise `s` in several ways:

| | |
|---|---|
| `s % [1,0,0]` | is the first marginal (projection) |
| `s % [0,1,0]` | is the second marginal |
| `s % [0,1,1]` | is the second and third marginal, *etc.* |

In general, for a state on the product of $n$ domains, a 'mask' of length $n$ is used with 0's and 1's, where a 1 at position $i$ tells that the $i$-th domain should be kept in the marginal.

Convex sums of states exist, like in `r1 * s1 + r2 * s2 + r3 * s3` for states `si` and numbers `ri` from $[0, 1]$ that add up to 1. Such convex sums exist in $n$-ary form.

Specific states are predefined, specifically for discrete/continuous/quantum probability, like uniform states, point states, Poisson, binomial, Gaussian, *etc.*

### Predicates

Predicates in probabilistic logic have the structure of an effect module [5]. On a fixed domain `D`, there are truth and falsity predicates `truth(D)` and `falsity(D)`, which are least and greatest element. Also, for each predicate `p` there is an orthosupplement (negation) `~p` and a rescaling `r*p` for a number `r` in the unit interval. There is a partially defined sum `p+q`, which is defined (as predicate) whenever this sum is below truth. Finally there is a sequential conjunction `p & q`. It is commutative in classical probability, but not in quantum probability.

For two predicates `p` on domain `D` and `q` on `E` there is also the parallel conjunction `p @ q` on the product of the underlying domains. This construction is often used for weakening, when either `p` or `q` is truth, in order to move a predicate to a bigger context.

### States and predicates

Given a state `s` and a predicate `p` on the same domain, one writes `s >= p` in EfProb for the validity of `p` in `s`. This yields a number in $[0, 1]$. Also one can write `s/p` for the result of conditioning `s` with `p`. The following version of Bayes' rule holds in general: `s/p >= q` equals `(s >= p & q) / (s >= p)`. In classical probability the order of conditioning is irrelevant — that is, `s/p/q` is the same as `s/q/p` — but in quantum probability the order does matter.

### Channels

A channel `c` is a 'probabilistic map' from a domain to a codomain, say from `D` to `E`. For a state `s` on `D`, one writes `c >> s` for the state on `E` obtained by state transformation. In the other direction, for a predicate `q` on `E` one obtains predicate `c << q` on `D` by predicate transformation. There is a basic law that tells that the probabilities `(c >> s) >= q` and `s >= (c << q)` are equal. There are operations $*$ and `@` for sequential and parallel composition of channels. They can be used as a basis for an elementary language for writing programs/protocols.

Channels are often used in Bayesian (backwards) learning (see [7]), in the form of an updated (revised) posterior state `s / (c << q)` of a prior state `s` in the light of evidence `q`.

The EfProb library supports more structure and more functions, for instance for random variables and Bayesian networks, expected values and (co)variance, and for disintegration. In the remainder of this note we sketch some examples, in order to give an impression of the possibilities. For more information we refer to the EfProb Manual [6].

## 3 Discrete probability

A discrete state/probability distribution $\omega$ on a finite domain $D$ is a probability mass function $\omega\colon D \to [0,1]$ that satisfies $\sum_{x\in D} \omega(x) = 1$. EfProb prints such distributions using 'ket' notation, as $\frac{1}{2}|a\rangle + \frac{1}{6}|b\rangle + \frac{1}{3}|c\rangle$, for $\{a,b,c\} \subseteq D$. A predicate on $D$ is a function $p\colon D \to [0,1]$. Validity $\omega \models p$ is defined as $\sum_{x\in D} \omega(x) \cdot p(x)$. If this number is non-zero the conditioned state $\omega/p\colon D \to [0,1]$ is given by $(\omega/p)(x) = \frac{\omega(x)\cdot p(x)}{\omega \models p}$. A channel $D \to E$ is a function from $D$ to distributions on $E$. It is a Kleisli map for the distribution monad that can be understood as a $D$-indexed collection of states on $E$. It is represented in EfProb/Python as a stochastic matrix. Then: $(c \gg \omega)(y) = \sum_x \omega(x) \cdot c(x)(y)$ and $(c \ll q)(x) = \sum_y c(x)(y) \cdot q(y)$.

We consider a beginner's example in Bayesian reasoning. The setting is given by some disease with a priori probability of 1%. There is a test for the disease with the following 'sensitivity'. If someone has the disease, then the test is 90% positive; but if someone does not have the disease, there is still a 5% chance that the test is positive. We formalise this situation in EfProb as a prior state and a channel:

```
>>> disease_dom = ['D', '~D']
>>> prior = flip(1/100, disease_dom)
>>> prior
0.01|D> + 0.99|~D>
>>> sensitivity = chan_from_states([flip(9/10), flip(1/20)], disease_dom)
```

We first use the channel to compute the probability that a test for an arbitrary person is positive. This is done via state transformation:

```
>>> sensitivity >> prior
0.0585|True> + 0.942|False>
```

Next we would like to learn the probability of having the disease after a positive test. This 'positive test' predicate is written as `yes_pred`. It is transformed into a predicate on the disease domain via the sensitivity channel, and then used to update the prior state below. We see that after a positive test the disease probability changes from 1% to 15%.

```
>>> posterior = prior / (sensitivity << yes_pred)
>>> posterior
0.154|D> + 0.846|~D>
```

## 4 Continuous probability

In general, a continuous state/probability distribution consists of a measurable space $(X, \Sigma)$ with a probability measure $\omega\colon \Sigma \to [0,1]$. In practice such measures are often given by a probability density function (pdf). This approach is followed in EfProb. Thus, a continuous state is given by a domain consisting of an $n$-dimensional cube $D$ of real numbers, possibly infinite, with a pdf $\omega\colon D \to \mathbb{R}_{\geq 0}$ satisfying $\int_D \omega(x)\,\mathrm{d}x = 1$. A predicate on $D$ is a (measurable) function $p\colon D \to [0,1]$. Validity $\omega \mathrel{>\!\!=} p$ is given by $\int_D \omega(x) \cdot p(x)\,\mathrm{d}x$. The updated state/pdf $\omega/p\colon D \to [0,1]$ sends $x \in D$ to the fraction $\frac{\omega(x)\cdot p(x)}{\omega \mathrel{>\!\!=} p}$, when the validity $\omega \mathrel{>\!\!=} p$ is non-zero. A channel $D \to E$ is given by a parameterised pdf $c\colon D \times E \to \mathbb{R}_{\geq 0}$, with $\int_E c(x,y)\,\mathrm{d}y = 1$ for each $x \in D$. Then $(c \ll \omega)(y) = \int_D \omega(x) \cdot c(x,y)\,\mathrm{d}x$ and $(c \gg q)(x) = \int_E c(x,y) \cdot q(y)\,\mathrm{d}y$.

We sketch an example that combines discrete and continuous probability. The setting is given by the capture and recapture methodology to estimate the size of a population in ecology. Imagine we are looking at a pond and we wish to learn the number of fish. We catch twenty of them, mark them, and throw them back. Subsequently we catch another twenty, and find out that five of them are marked. What do we learn about the number of fish?

The number of fish in the pond must be at least 20. Let's assume the maximal number is 300. We thus take the interval $[20, 300] \subseteq \mathbb{R}$ as domain. We start from the uniform distribution since we don't assume any prior knowledge about the distribution of fish.

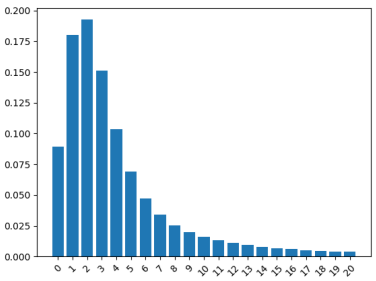
```
>>> fish_dom = R(20,300)
>>> prior = uniform_state(fish_dom)
>>> prior.expectation()
160.0
```

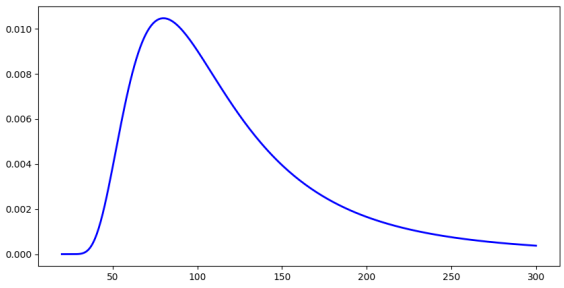
The method `expectation()` computes the mean value of the distribution.

We now assume that 20 of the fish in the pond are marked. We compute for each $x \in [20, 300]$ in the fish domain the probability of finding 5 marked fish when 20 of them are caught. For this we use the binomial distribution with parameters $N = 20$ and probability $p = \frac{x}{20}$. This is incorporated in the following channel, from the fish domain to the booleans.

```
>>> c = chan_fromklmap(lambda x: binomial(20, 20/x), fish_dom, range(21))
>>> (c >> prior).plot()
```

The latter plot command produces the (discrete) distribution on the left below. It gives the probability for each $k \in \{0, 1, \ldots, 20\}$ of catching $k$ marked fish, in the prior uniform state.



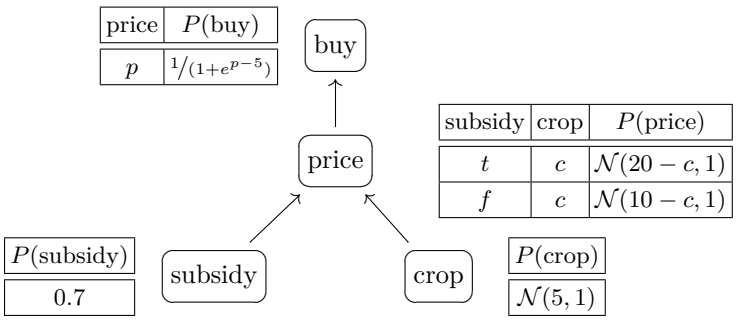

The plot on the right describes the posterior state that is obtained by updating the prior with the information that five marked fish have been found. The latter is expressed via a transformed predicate, as below.

```
>>> posterior = prior / (chan << point_pred(5, range(21)))
```

```
>>> posterior.plot()
>>> posterior.expectation()
116.491929836
```

The expected number of fish in the pond after recapture is calculated as 116.

We include another illustration, namely a *hybrid* Bayesian network, in which discrete and continuous probability are combined. Consider the following network from [2], with two discrete nodes (namely subsidy and buy) and two continuous ones (crop and price). The normal price distribution depends continuously on the crop, which is used as input to the its mean, in a way that depends whether subsidy is given. Whether or not the product will be bought is described as a biased coin, with bias parameter given by a 'sigmoid' function.




The conditional probability tables describe how nodes depend on each other. We write $\mathcal{N}(\mu, \sigma)$ for the normal (Gaussian) distribution with mean $\mu$ and standard deviation $\sigma$.

This network is modeled in EfProb in the following way. First, the subsidy and corp distributions form a joint prior state:

```
>>> subsidy = flip(0.7)
>>> dom = R(0,20)
>>> crop = gaussian_state(5,1,dom)
>>> prior = subsidy @ crop
```

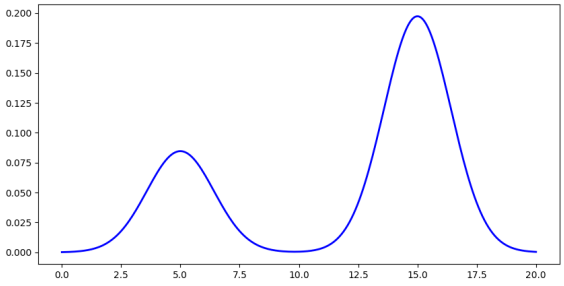
As indicated, the domain of the crop distribution is the real interval $[0, 20]$. The above two conditional probability for price and buy are translated into channels:

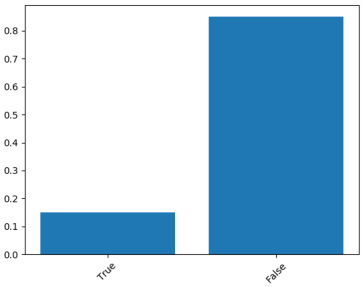
```
>>> price = chan_fromklmap(lambda p,c: gaussian_state(20-c, 1, dom) if p
...                        else gaussian_state(10-c, 1, dom),
...                        [bool_dom, dom], dom)
>>> buy = chan_fromklmap(lambda p: flip(1/(1+exp(p-5))), dom, bool_dom)
```

We see that the translation from the Bayesian network to the EfProb code is rather straightforward. We can now do 'forward' calculations to obtain distributions for price and buy by state transformation:

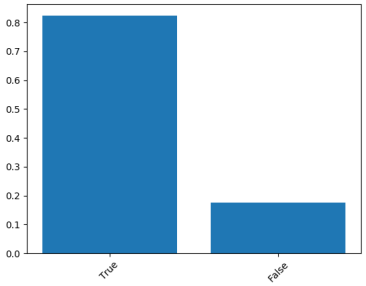
```
>>> p = price >> prior
>>> p.expectation(), p.variance()
11.9997758228 22.9911228888
>>> b = buy >> p
>>> b >= yes_pred
0.150080181436
```

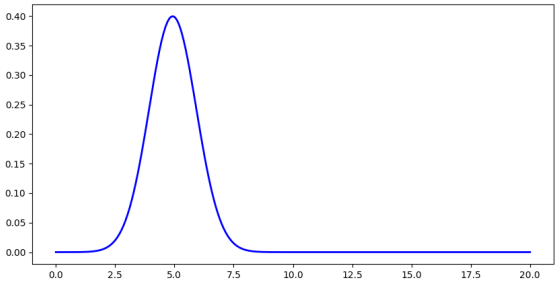
Computing these numbers takes a few seconds, on an ordinary laptop. These values are (very close to) the ones reported in [2]. The plots of the computed prior price and buy distributions (`p` and `b`) are given below. We see that the price distribution is a convex sum of two normal distributions.




We can also do 'backward' reasoning when we observe, for instance, that the product is not bought. The resulting posterior subsidy and crop distributions become:




We see that the subsidy probability increases to 82%; the crop distribution looks unchanged, but its mean actually drops from 5 to 4.9. This update prior is produced via predicate transformation, using a sequential composition channel `buy * price` in:

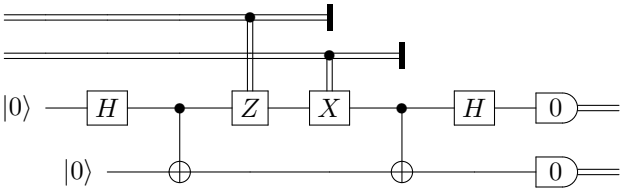
```
>>> posterior = prior / ((buy * price) << no_pred)
>>> (posterior % [1,0]).plot()
>>> (posterior % [0,1]).plot()
```

The two marginals of the posterior state, for subsidy and crop, are plotted.

## 5 Quantum probability

A quantum state of dimension $n$ is an $n \times n$ complex matrix $\omega$ that is positive and has trace one: $\mathrm{tr}(\omega) = 1$. A predicate $p$ is a positive matrix below the identity matrix: $0 \leq p \leq \mathrm{id}$. The validity $\omega \mathrel{>\!\!=} p$ is defined via the Born rule as $\mathrm{tr}(\omega\, p)$. The conditioned state $\omega/p$ is $\frac{\sqrt{p}\,\omega\,\sqrt{p}}{\omega \mathrel{>\!\!=} p}$. A channel $n \to m$ is a positive unitary linear map from $m \times m$ matrices to $n \times n$ matrices. We refer to [9] for more background information.

We briefly describe the so-called superdense coding protocol that is represented as a circuit on the right, using notation like in Quipper [4] and QWire [10]. In this protocol Alice transfers two classical bits to Bob via two entangled qubits — in the



form of a Bell state that is produced in the lower-left part of the circuit — where Alice possesses one qubit and Bob the other. The EfProb code for Alice, Bob, and the whole protocol is described via sequential $*$ and parallel composition `@` of channels:

```
>>> alice = (discard(2) @ idn(2)) * ccontrol(x_chan) \
...         * (idn(2) @ discard(2) @ idn(2)) \
...         * (idn(2) @ ccontrol(z_chan)) * (swap @ idn(2))
>>> bob = (meas0 @ meas0) * (hadamard @ idn(2)) * cnot
>>> sdc = bob * (alice @ idn(2)) * (idn(2,2) @ bell00.as_chan())
```

We can run this protocol with two classical (probabilistic) states as input:

```
>>> s = random_probabilistic_state(2)
>>> t = random_probabilistic_state(2)
>>> s
[[ 0.89708576  0.        ]
 [ 0.          0.10291424]]
>>> (sdc >> s @ t) % [1,0]
[[ 0.89708576+0.j  0.00000000+0.j]
 [ 0.00000000+0.j  0.10291424+0.j]]
```

Similarly the second marginal of `sdc >> s @ t` yields the original second input `t`.

## References


1. K. Cho, B. Jacobs, A. Westerbaan, and B. Westerbaan. An introduction to effectus theory. Preprint, 2015. arXiv:1512.05813 [cs.LO].
2. B. Cobb and P. Shenoy. Inference in hybrid Bayesian networks with mixtures of truncated exponentials. *Int. J. Approx. Reasoning*, 41(3):257–286, 2006.
3. N. Goodman, V. Mansinghka, D. Roy, K. Bonawitz, and J. Tenenbaum. Church: a language for generative models. In *Uncertainty in Artificial Intelligence*, 2008.
4. A. Green., P. LeF. Lumsdaine, N. Ross, P. Selinger, and B. Valiron. Quipper: A scalable quantum programming language. In *Programming Language Design and Implementation*, 2013.
5. B. Jacobs. New directions in categorical logic, for classical, probabilistic and quantum logic. *Logical Methods in Comp. Sci.*, 11(3):1–76, 2015. `doi:10.2168/LMCS-11(3:24)2015`.
6. B. Jacobs and K. Cho. EfProb user manual. See `efprob.cs.ru.nl`, 2017.
7. B. Jacobs and F. Zanasi. A predicate/state transformer semantics for Bayesian learning. In L. Birkedal, editor, *Math. Found. of Programming Semantics*, number 325 in Elect. Notes in Theor. Comp. Sci., pages 185–200. Elsevier, Amsterdam, 2016.
8. B. Milch, B. Marthi, S. Russell, D. Sontag, D. Ong, and A. Kolobov. BLOG: Probabilistic models with unknown objects. In L. Getoor and B. Taskar, editors, *Introduction to Statistical Relational Learning*. MIT Press, 2007.
9. M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information*. Cambridge Univ. Press, 2000.
10. J. Paykin, R. Rand, and S. Zdancewic. QWIRE: A core language for quantum circuits. In *Princ. of Programming Languages*, pages 846–858. ACM Press, 2017.
11. F. Wood, J. W. van de Meent, and V. Mansinghka. A new approach to probabilistic programming inference. In *Artificial Intelligence and Statistics*, 2014.