Kleisli Semantics for Conditioning in Probabilistic Programming[*]

Kenta Cho and Bart Jacobs

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

Abstract

This paper investigates the categorical semantics of conditioning in probabilistic programming using monads. It concentrates on terminating probabilistic programs, and presents two styles of semantics: one 'one-deficit' approach via the lift monad $(-)+1$, and another 'monoid' approach using the monad $2 \times (-)$ induced by the monoid 2. In both cases, a correctness result is proven, showing that it suffices to normalise at the end of a program. The two semantics are shown to be equivalent. Secondly the paper briefly, and non-exhaustively, investigates the extension of the monadic semantics to possibly non-terminating programs. This requires a new 'liberal' interpretation of conditioning of substates.

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages

Keywords and phrases conditioning, probabilistic programming, monad, Kleisli category

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

Probabilistic programming with Bayesian conditioning is an active research area, see *e.g.* [1, 3, 7–12]. Here we concentrate on a categorical semantics, in Kleisli categories, for finite, discrete probability. We investigate several alternative forms of semantics, concentrating on conditioning, by involving the 'lift' monad and the 'monoid' monad.

The standard way to model partial programs uses the lift monad $(-)+1$, for instance on the category **Set** of sets and functions. It adds a new element $*$ from the one-element (final) set $1 = \{*\}$ to an arbitrary set, in $X \mapsto X+1$. A partial program $X \to Y$ can then be written simply as a map $X \to Y$ in the Kleisli category $\mathcal{K}\ell((-)+1)$ of the lift monad, that is, as an ordinary function $X \to Y+1$. Sequential composition of such partial programs is given by Kleisli composition in $\mathcal{K}\ell((-)+1)$, and parallel composition is given by the tensor product $f \otimes g$ of two maps f, g in $\mathcal{K}\ell((-)+1)$.

This lift construction is quite flexible, and can also be used in the context of probabilistic programming. For finite discrete probability we use the distribution monad $\mathcal{D}$. The lift monad can be defined on the Kleisli category $\mathcal{K}\ell(\mathcal{D})$. We can then again form the Kleisli category of lift, which we shall write as $\mathcal{K}\ell_{\mathcal{D}}((-)+1)$, with a subscript '$\mathcal{D}$', in order to prevent confusion with the Kleisli category of the lift monad on the underlying category **Set**. It is a basic fact that distributions in $\mathcal{D}(Y+1)$ can be identified with subdistributions on Y.

What we have sketched so far is the use of the lift monad for partiality, in the context of probabilistic programming. This article will introduce another usage of the lift monad, for

[*] 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.

normalisation and conditioning. In fact, we shall introduce a second monad for normalisation and conditioning, and describe its relationship to lift. We briefly describe how this works.

- A subdistribution $\omega \in \mathcal{D}(Y+1)$ can be *normalised*, if it is non-zero, via rescaling. If we describe $\omega\colon Y \to [0,1]$ as a function with finite support and $\|\omega\| := \sum_y \omega(y) \leq 1$, then its normalisation $\mathrm{nrm}(\omega)\colon Y \to [0,1]$ is given by $\mathrm{nrm}(\omega)(y) = \frac{\omega(y)}{\|\omega\|}$. The value $1 - \|\omega\|$ is sometimes called the 'one-deficit' of ω. We shall refer to this as the one-deficit approach.
- The two-element set $2 = \{\mathrm{yes}, \mathrm{no}\}$ is a commutative monoid with conjunction. The mapping $X \mapsto 2 \times X$ is a monad on **Set**, and also on the Kleisli category $\mathcal{K}\ell(\mathcal{D})$ of the distribution monad. We shall work in its Kleisli category, written as $\mathcal{K}\ell_{\mathcal{D}}(2 \times (-))$. There is a map $2 \times Y \cong Y + Y \to Y + 1$ to lift, which is actually a map of monads. A distribution $\omega \in \mathcal{D}(2 \times Y)$ can also be normalised, namely via conditioning and marginalisation, written as $\mathsf{M}_2(\omega|_{\mathrm{yes}\otimes\mathbf{1}})$. Details will be given below. We refer to this description as the monoid approach.

This paper will (abstractly) develop the semantics of probabilistic programming with conditioning, both for the one-deficit approach and for the monoid approach, and will relate them. It will be shown formally, for both approaches, that normalisation can be postponed to the end of the program, since it propagates through the whole program. Both approaches have been implemented using the EfProb library[1] for probability calculations. EfProb forms an embedded language in Python. The implementations are illustrated in a standard Bayesian example, for finding out the number of fish in a pond by looking at a sample.

These two 'one-deficit' and 'monoid' approaches in Sections 4 and 5 form the technical core and main contribution of the paper. In Section 7 we look at extensions of probabilistic programming with non-termination, via an abort statement, that fails with certainty. There are theoretically several ways to combine non-termination with the 'one-deficit' and 'monoid' approach, namely as:

1. $(-) + 1 + 1$, where the inner lift is used for partiality, and the outer one for conditioning;
2. $\big(2 \times (-)\big) + 1$
3. $2 \times \big((-) + 1\big)$.

In this paper we do not give an exhaustive analysis. The first option is not investigated here, because it is somewhat confusing, using lift in a double role. The second approach looks attractive, because it can be implemented relatively easily in EfProb, using subdistributions. However, we show that it does not satisfy the required mathematical properties, and gives the wrong outcome in an example. Hence, in the end, the third option looks the most attractive. We illustrate by hand that it gives the right outcomes in two examples. However, we do not have an implementation for it.

The term 'program' is used in a rather loose manner. We do not formally define a syntax and hope that the reader will accept our hints about how our semantical constructs correspond to programming constructs. We provide several examples, which should be self-explanatory, given a basic level of familiarity with probabilistic programming [3]. We do not consider loops or recursion in the current setting, unlike *e.g.* [7, 10]. They lead to interesting theoretical questions, but their relevance for practical examples in Bayesian reasoning is unclear to us. The same may be said, by the way, about non-termination via an abort statement.

The two semantics for probabilistic programs given in [11] and [12] resemble our 'one-deficit' and 'monoid' approach: they respectively use s-finite kernels $X \rightsquigarrow Y$ and measurable

[1] Developed by the authors, see `efprob.cs.ru.nl`.

functions $X \to \mathcal{G}(\mathbb{R}_{\geq 0} \times Y)$. The formal relationship is to be investigated. One difference is our emphasis on monads $(-)+1$ and $2 \times (-)$ and on the effectus theoretic perspective, via predicates p and conditional state $\omega|_p$.

2 Monads

We start with some categorical generalities that capture the setting in which we work. This section provides some background information, but is not essential for what follows.

Let $T = (T, \eta, \mu)$ be a monad on a symmetric monoidal category $\mathbf{C} = (\mathbf{C}, I, \otimes)$. We assume that this monad is *commutative* (or *monoidal*): it comes with natural 'double strength' maps $\mathrm{dst}\colon T(X) \otimes T(Y) \to T(X \otimes Y)$ which make certain standard diagrams commute, see *e.g.* [5]. We write $\mathcal{K}\ell(T)$ for the Kleisli category of T. It is then also symmetric monoidal.

Let $\mathbf{C}$ have coproducts $+$ and a final object 1. It gives rise to the 'lift' monad $(-)+1\colon \mathbf{C} \to \mathbf{C}$. There is always a distributive law of monads with components $T(X)+1 \to T(X+1)$. This means that the lift monad lifts to the Kleisli category.

Let $M \in \mathbf{C}$ be a commutative monoid, via maps $u\colon I \to M$ and $m\colon M \otimes M \to M$. The functor $M \otimes (-)\colon \mathbf{C} \to \mathbf{C}$ is then a monad, which is commutative since M is commutative as a monoid. Moreover, the strength map $\mathrm{st}\colon M \otimes T(X) \to T(M \otimes X)$, given by $\mathrm{st} = \mathrm{dst} \circ (\mathrm{id} \otimes \eta)$ is a distributive law of monads. As a result, tensoring with M lifts to a monad $M \otimes (-)\colon \mathcal{K}\ell(T) \to \mathcal{K}\ell(T)$ on the Kleisli category of T. We thus have the following situation, with two (lifted) monads and their Kleisli categories.

$$\begin{tikzcd}
\mathcal{K}\ell_T((-)+1) \arrow[dr] & & \mathcal{K}\ell_T(M \otimes (-)) \arrow[dl] \\
& \mathcal{K}\ell(T) \arrow[loop left, "(-)+1"] \arrow[loop right, "M\otimes(-)"] \arrow[d] & \\
& \mathbf{C} \arrow[loop left, "(-)+1"] \arrow[loop right, "M\otimes(-)"] &
\end{tikzcd} \tag{1}$$

We use a subscript T for the upper Kleisli categories, in order to prevent confusion with the Kleisli categories $\mathcal{K}\ell((-)+1)$ and $\mathcal{K}\ell(M \otimes (-))$ of the monads $(-)+1$ and $M \otimes (-)$ on $\mathbf{C}$.

These Kleisli categories $\mathcal{K}\ell_T((-)+1)$ and $\mathcal{K}\ell_T(M \otimes (-))$ are then also symmetric monoidal, where for the first case we need to assume that tensors distribute over coproducts. Below we briefly describe sequential and parallel composition, for Kleisli maps $X \to Y$, $Y \to Z$ and $A \to B$. Since programs will be interpreted in these categories later on, we will write ; and $\otimes$ for sequential and parallel composition in these case.

We start with $\mathcal{K}\ell_T((-)+1)$, using composites in the underlying category $\mathbf{C}$.

$$\begin{tikzcd}
X \arrow[d, "f"'] \\
T(Y+1) \arrow[d, "T(g+\mathrm{id})"'] \\
T(T(Z+1)+1) \arrow[d, "T(\mathrm{st})"'] \\
T^2((Z+1)+1) \arrow[d, "\mu"'] \\
T((Z+1)+1) \arrow[d, "T([\mathrm{id},\kappa_2])"'] \\
T(Z+1)
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
X \otimes A \arrow[d, "f \otimes g"] \\
T(Y+1) \otimes T(B+1) \arrow[d, "\mathrm{dst}"] \\
T\big((Y+1) \otimes (B+1)\big) \arrow[d, equal, "\wr"] \\
T\big((Y \otimes B) + (Y \otimes 1) + (1 \otimes B) + (1 \otimes 1)\big) \arrow[d, "{[\kappa_1, \kappa_2 \circ !, \kappa_2 \circ !, \kappa_2 \circ !]}"] \\
T\big((Y \otimes B) + 1\big)
\end{tikzcd}$$

Sequential and parallel composition in $\mathcal{K}\ell_T(M \otimes (-))$ takes the following form.

$$\begin{tikzcd}
X \arrow[d, "f"'] \\
T(M \otimes Y) \arrow[d, "T(\mathrm{id}\otimes g)"'] \\
T(M \otimes T(M \otimes Z)) \arrow[d, "\mathrm{st}"'] \\
T^2(M \otimes (M \otimes Z)) \arrow[d, equal, "\wr"'] \\
T^2((M \otimes M) \otimes Z) \arrow[d, "\mu"'] \\
T((M \otimes M) \otimes Z) \arrow[d, "T(m\otimes\mathrm{id})"'] \\
T(M \otimes Z)
\end{tikzcd}
\qquad
\begin{tikzcd}
X \otimes A \arrow[d, "f\otimes g"] \\
T(M \otimes Y) \otimes T(M \otimes B) \arrow[d, "\mathrm{dst}"] \\
T\big((M \otimes Y) \otimes (M \otimes B)\big) \arrow[d, equal, "\wr"] \\
T\big((M \otimes M) \otimes (Y \otimes B)\big) \arrow[d, "T(m\otimes\mathrm{id})"] \\
T\big(M \otimes (Y \otimes B)\big)
\end{tikzcd}$$

2.1 Distribution monads

We shall be using the above set-up for $\mathbf{C} = \mathbf{Set}$ and commutative monoid $2 = \{\mathrm{yes}, \mathrm{no}\}$ with conjunction $\wedge$ and yes $\in 2$ as neutral element. Two monads will be considered, namely the (finite probability) distribution monad $\mathcal{D}$, and also the subdistribution monad $\mathcal{D}_{\leq 1}$. We briefly describe them both.

A *distribution* on a set X is given by a probability mass function $\omega\colon X \to [0,1]$ with finite support $\mathrm{supp}(\omega) = \{x \mid \omega \neq 0\}$ and $\sum_x \omega(x) = 1$. For a *subdistribution* the latter condition is relaxed to $\sum_x \omega(x) \leq 1$. Sometimes we describe a (sub)distribution via the 'ket' notation as a formal sum $\sum_x \omega(x)|x\rangle$.

We use the term '(sub)state' as alternative for '(sub)distribution'. Also we call a distribution a 'proper' distribution if we wish to emphasise that it is a 'non-sub' distribution.

We write $\mathcal{D}(X)$ for the set of (proper) distributions on X, and $\mathcal{D}_{\leq 1}(X)$ for the set of subdistributions. It is well-known that both $\mathcal{D}$ and $\mathcal{D}_{\leq 1}$ are commutative monads on **Set**, see *e.g.* [5]. It is not hard to see that $\mathcal{D}_{\leq 1}(X) \cong \mathcal{D}(X + 1)$. Both $\mathcal{D}(X)$ and $\mathcal{D}_{\leq 1}(X)$ are closed under convex sums, often written as $\omega +_r \rho = r \cdot \omega + (1 - r) \cdot \rho$, for $r \in [0, 1]$.

A (sub)distribution ω on a product $X \times Y$ is often called a 'joint' distribution. For such an ω we can define the marginal distributions $\mathsf{M}_1(\omega)$ on X and and $\mathsf{M}_2(\omega)$ on Y as:

$$\mathsf{M}_1(\omega)(x) := \textstyle\sum_y \omega(x, y) \qquad \text{and} \qquad \mathsf{M}_2(\omega)(y) := \textstyle\sum_x \omega(x, y).$$

Given two distributions $\omega \in \mathcal{D}(X)$ and $\rho \in \mathcal{D}(Y)$ we write $\omega \otimes \rho = \mathrm{dst}(\omega, \rho) \in \mathcal{D}(X \times Y)$ for the (joint) product state. It is given by $(\omega \otimes \rho)(x, y) = \omega(x) \cdot \rho(y)$. The same construction will be used for subdistributions. One has $\mathsf{M}_1(\omega \otimes \rho) = \omega$ and $\mathsf{M}_2(\omega \otimes \rho) = \rho$.

For a Kleisli map, also called a 'channel', $f\colon X \to \mathcal{D}(Y)$ and a state $\omega \in \mathcal{D}(X)$ we write $f_*(\omega) \in \mathcal{D}(Y)$ for the transformed state defined by $f_*(\omega)(y) = \sum_x \omega(x) \cdot f(x)(y)$. This 'state transformer' f_* will also be used for subdistributions.

3 Predicates, validity and conditioning

This section first recalls the basic notions of validity of a predicate in a state, and of conditioning (updating, revising) a state, given a predicate. In the first part the 'standard' approach from effectus theory is given, together with an example illustrating its use in probabilistic programming. In a separate subsection a novel 'liberal' variation is described for substates.

In the current context, a *predicate* on a set X is a function $p\colon X \to [0,1]$, which may be identified with a Kleisli map $X \to \mathcal{D}(2)$. We write $\mathbf{1}, \mathbf{0} \in [0,1]^X$ for the constant-1 and constant-0 predicates, which are used as truth and falsity. The orthosupplement $p^\perp \in [0,1]^X$ of predicate $p \in [0,1]^X$ is defined as $(p^\perp)(x) = 1 - p(x)$. Obviously, $p^{\perp\perp} = p$. For two predicates $p \in [0,1]^X, q \in [0,1]^Y$ we write $p \otimes q \in [0,1]^{X \times Y}$ for the parallel conjunction, given by $(p \otimes q)(x,y) = p(x) \cdot q(y)$. Conjunction with truth $p \otimes \mathbf{1}$ and $\mathbf{1} \otimes q$ is *weakening*, that is, moving a predicate to a bigger context.

The *validity* $\omega \models p$, for $\omega \in \mathcal{D}(X)$ and $p \in [0,1]^X$, is defined as the number in $[0,1]$ given on the left below. If this number is non-zero, the conditioned state $\omega|_p \in \mathcal{D}(X)$ can be defined as on the right.

$$\omega \models p \coloneqq \textstyle\sum_x \omega(x) \cdot p(x) \qquad\qquad \omega|_p(x) \coloneqq \frac{\omega(x) \cdot p(x)}{\omega \models p}. \tag{2}$$

Conditioning satisfies some basic rules (see [6]) like:

$$\omega|_{\mathbf{1}} = \omega \qquad (\omega \otimes \rho)|_{p \otimes q} = (\omega|_p) \otimes (\rho|_q) \qquad (\omega|_p)|_q = \omega|_{p \& q},$$

where $(p \& q)(x) = p(x) \cdot q(x)$, that is, $p \& q = (p \otimes q) \circ \Delta$. Moreover, Bayes' law holds:

$$\omega|_p \models q = \frac{\omega \models p \& q}{\omega \models p}. \tag{3}$$

► **Example 1.** Consider the following program fragment.

```
{x := 0} +_{1/3} {x := 1};
{y := 0} +_{1/2} {y := 1};
observe (x+y = 1);
```

Semantically one can describe the state after the first two lines as a product $\omega_x \otimes \omega_y \in \mathcal{D}(\{0,1\} \times \{0,1\})$ where $\omega_x = \frac{1}{3}|0\rangle + \frac{2}{3}|1\rangle$ and $\omega_y = \frac{1}{2}|0\rangle + \frac{1}{2}|1\rangle$. The predicate $p \in [0,1]^{\{0,1\} \times \{0,1\}}$ used in the observe line is given by $p(0,0) = p(1,1) = 0$ and $p(0,1) = p(1,0) = 1$. The outcome of this program is described by conditioning:

$$\begin{aligned} \omega_x \otimes \omega_y &= \tfrac{1}{6}|00\rangle + \tfrac{1}{6}|01\rangle + \tfrac{1}{3}|10\rangle + \tfrac{1}{3}|11\rangle \\ \omega_x \otimes \omega_y \models p &= \tfrac{1}{6}p(0,0) + \tfrac{1}{6}p(0,1) + \tfrac{1}{3}p(1,0) + \tfrac{1}{3}p(1,1) = \tfrac{1}{2} \\ (\omega_x \otimes \omega_y)|_p &= \tfrac{1/6}{1/2}|01\rangle + \tfrac{1/3}{1/2}|10\rangle = \tfrac{1}{3}|01\rangle + \tfrac{2}{3}|10\rangle. \end{aligned}$$

These `observe` statements will be interpreted either as 'instrument', for the monoid monad $2 \times (-)$, or as 'assert' maps, for the lift monad $(-)+1$. Instruments and asserts belong to the standard machinery of effectus theory [2,4]. Here we describe them in more concrete form, for discrete probability. Each predicate gives rise to both an instrument and an assert map, as in (4) below. These maps form morphisms in the Kleisli categories $\mathcal{K}\ell_{\mathcal{D}}(2 \times (-))$ and $\mathcal{K}\ell_{\mathcal{D}}((-)+1)$ from (1). They are defined as the following composites in $\mathcal{K}\ell(\mathcal{D})$.

$$\begin{aligned} \mathsf{instr}_p &\coloneqq \Big(X \xrightarrow{\Delta} X \times X \xrightarrow{p \times \mathrm{id}} 2 \times X\Big) \\ \mathsf{asrt}_p &\coloneqq \Big(X \xrightarrow{\mathsf{instr}_p} 2 \times X \cong X + X \xrightarrow{\mathrm{id}+!} X + 1\Big). \end{aligned} \tag{4}$$

Concretely, these maps are described by:

$$\mathsf{instr}_p(x) = p(x)|\mathrm{yes}, x\rangle + (1 - p(x))|\mathrm{no}, x\rangle \qquad \text{and} \qquad \mathsf{asrt}_p(x) = p(x)|x\rangle.$$

These instrument and assert maps satisfy:

$$\begin{array}{rclcrcl} \mathsf{instr}_{\mathbf{1}} & = & \mathrm{id} & \qquad & \mathsf{instr}_{p \otimes q} & = & \mathsf{instr}_p \otimes \mathsf{instr}_q \\ \mathsf{asrt}_{\mathbf{1}} & = & \mathrm{id} & & \mathsf{asrt}_{p \otimes q} & = & \mathsf{asrt}_p \otimes \mathsf{asrt}_q. \end{array} \tag{5}$$

The identity maps and tensors on the right-hand-side of the equality signs live in the Kleisli categories $\mathcal{K}\ell_{\mathcal{D}}(2 \times (-))$ and $\mathcal{K}\ell_{\mathcal{D}}((-) + 1)$.

3.1 Liberal validity and conditioning

The above descriptions $\omega \models p$ and $\omega|_p$ in (2) assume that ω is a proper distribution. What if ω is a subdistribution? We could apply the same definitions, but then we see that $\omega|_p$ automatically becomes proper. We would like to take non-definedness into account. This leads to the alternative approach that we sketch below. We call it 'liberal' since in weakest precondition semantics the word 'liberal' is also used for this special treatment of undefinedness. Our approach is 'new' in the sense that, as far as we know, it has not been formulated at this general level, with substates and predicates, including a liberal version of Bayes' rule.

A non-zero subdistribution $\omega \in \mathcal{D}_{\leq 1}(X)$ can be turned into a (proper) distribution $\mathrm{nrm}(\omega) \in \mathcal{D}(X)$ via *normalisation.* One takes:

$$\|\omega\| := \textstyle\sum_x \omega(x) \qquad \text{and then} \qquad \mathrm{nrm}(\omega)(x) := \frac{\omega(x)}{\|\omega\|}. \tag{6}$$

The difference $\|\omega\|^{\perp} = 1 - \|\omega\|$ is sometimes called the 'one-deficit' of ω. It is describes the extent to which ω is undefined, as a number in the unit interval $[0, 1]$.

Subdistributions, unlike proper distributions, are closed under multiplication with a number $r \geq 0$. This operation satisfies $\|r \cdot \omega\| = r \cdot \|\omega\|$. This shows that we need to impose the requirement that $r \leq \frac{1}{\|\omega\|}$. State transformation preserves such scalar multiplication: $f_*(r \cdot \omega) = r \cdot f_*(\omega)$. More generally, subdistributions can be multiplied pointwise with a predicate. We write $p \cdot \omega$ for the subdistribution $(p \cdot \omega)(x) = p(x) \cdot \omega(x)$.

Let $\omega \in \mathcal{D}_{\leq 1}(X)$ now be a subdistribution and $p \in [0, 1]^X$ be a predicate. We define 'liberal' versions of validity $\omega \models^{\ell} p$ and conditioning $\omega|^{\ell}_p$ as:

$$\omega \models^{\ell} p := \big(\textstyle\sum_x \omega(x) \cdot p(x)\big) + \|\omega\|^{\perp} \qquad\qquad \omega|^{\ell}_p(x) := \frac{\omega(x) \cdot p(x)}{\omega \models^{\ell} p}. \tag{7}$$

We note that if ω happens to be a proper distribution, then $\|\omega\| = 1$, so that these definitions reduce to the earlier ones (2). Hence we could drop the distinction sub/proper and use the liberal versions everywhere. But we prefer to keep the distinction for conceptual clarity.

These liberal versions of validity and conditioning will be used later on, in Section 7. For now we show that they satisfy a liberal version of Bayes' rule (3).

▶ **Lemma 2.** *For $\omega \in \mathcal{D}_{\leq 1}(X)$ and $p, q \in [0, 1]^X$ 'liberal Bayes' holds:*

$$\omega|^{\ell}_p \models q = \frac{\omega \models^{\ell} p \,\&\, q}{\omega \models^{\ell} p}.$$

Proof.

$$\begin{aligned}
\omega|^{\ell}_{p} \models^{\ell} q &\overset{(7)}{=} \textstyle\sum_x (\omega|^{\ell}_{p}(x) \cdot q(x)) + (1 - \|\omega|^{\ell}_{p}\|) \\
&= \sum_x \frac{\omega(x)\cdot p(x)}{\omega \models^{\ell} p} \cdot q(x) + (1 - \sum_x \frac{\omega(x)\cdot p(x)}{\omega \models^{\ell} p}) \\
&= \frac{\sum_x \omega(x)\cdot p(x)\cdot q(x)}{\omega \models^{\ell} p} + \frac{\omega \models^{\ell} p - \sum_x \omega(x)\cdot p(x)}{\omega \models^{\ell} p} \\
&\overset{(7)}{=} \frac{\sum_x \omega(x)\cdot (p \,\&\, q)(x)}{\omega \models^{\ell} p} + \frac{\|\omega\|^{\perp}}{\omega \models^{\ell} p} \overset{(7)}{=} \frac{\omega \models^{\ell} p \,\&\, q}{\omega \models^{\ell} p}.
\end{aligned}$$

◀

4 The one-deficit interpretation

In this section we use the Kleisli category $\mathcal{K}\ell_{\mathcal{D}}((-)+1)$ of the lift monad $(-)+1$ on $\mathcal{K}\ell(\mathcal{D})$ for program semantics. Its maps $X \to \mathcal{D}(Y+1)$ will be described in the form $X \to \mathcal{D}_{\le 1}(Y)$, producing subdistributions. At this stage partiality comes from the weights introduced by predicates, and not in the traditional way from 'abort' or from endless loops.

▶ **Definition 3.** For a Kleisli map $c\colon X \to \mathcal{D}_{\le 1}(Y)$, seen as a program, and a state $\omega \in \mathcal{D}(X)$, we write $c \Downarrow \omega$ for the (final) state after running program c in the (initial) state ω.

$$c \Downarrow \omega := \mathrm{nrm}\big(c_*(\omega)\big) \in \mathcal{D}(Y).$$

Thus, a 'run' applies the program c to the state ω via state transformation, giving a subdistribution $c_*(\omega) \in \mathcal{D}_{\le 1}(Y)$; subsequently, this subdistribution is normalised to a proper distribution. This 'run' is a partial operation, since the state $c_*(\omega)$ may be zero.

More concretely, $c \Downarrow \omega \in \mathcal{D}(Y)$ is the distribution with probability at $y \in Y$ given by:

$$\big(c \Downarrow \omega\big)(y) = \mathrm{nrm}\big(c_*(\omega)\big)(y) = \frac{c_*(\omega)(y)}{\|c_*(\omega)\|} = \frac{\sum_x c(x)(y)\cdot\omega(x)}{\sum_{x,y} c(x)(y)\cdot\omega(x)}. \tag{8}$$

In a next step we apply run to a sequence of program statements and show how we can go through the sequence iteratively. We consider the following program statements, for observe, if-then-else, and embed (including discard and assign). They will be closed under sequential and parallel composition ; and $\otimes$.

1. Observation of a predicate p on X is translated into an asrt_p endomap $X \to X$, that is, into a Kleisli map $\mathsf{asrt}_p\colon X \to \mathcal{D}_{\le 1}(X)$.
2. The statement $\mathsf{if}\ p\ \mathsf{then}\ f\ \mathsf{else}\ g\colon X \to \mathcal{D}_{\le 1}(Y)$, for $p \in [0,1]^X$ and $f, g\colon X \to \mathcal{D}_{\le 1}(Y)$, is defined as composite:

   $$\mathsf{if}\ p\ \mathsf{then}\ f\ \mathsf{else}\ g := \Big(X \xrightarrow{\mathsf{instr}_p} 2 \times X \cong X + X \xrightarrow{[f,g]} Y\Big).$$

   Then $(\mathsf{if}\ p\ \mathsf{then}\ f\ \mathsf{else}\ g)(x)(y) = p(x)\cdot f(x)(y) + (1-p(x))\cdot g(x)(y)$. As special case this construction includes a convex combination of f, g, namely when p is a constant predicate, that is, a scalar in the unit interval $[0,1]$.
3. The embedding statement $\mathsf{emb}(h)\colon X \to \mathcal{D}_{\le 1}(Y)$, for a 'pure' map $h\colon X \to \mathcal{D}(Y)$ in $\mathcal{K}\ell(\mathcal{D})$ is simply the inclusion. We make two special cases of this construction explicit.
   a. Embedding can be applied in particular to 'discard' $X \to 1$ or to 'projection' maps $X_1 \times X_2 \to X_i$, with which the context can be reduced, that is, with which variables can be removed.

b. Assignment is included as an embedding of a pure map. We only describe assignment to new variables, but assignment to existing variables can be done in a similar manner. Let X describe the current state. For a pure map $v\colon X \to \mathcal{D}(Y)$ we would like to extend the state X to $X \times Y$ by assigning a value sampled from a distribution to a new variable of type Y. This could be described as $\mathsf{new}\ y \leftarrow v(x)$, where v is a collection of proper distributions on Y, indexed by the existing state X. This leads to a 'graph' Kleisli map that extends the state from X to $X \times Y$.

$$\mathsf{assign}(v) \;\coloneqq\; \Big(X \xrightarrow{\Delta} X \times X \xrightarrow{\mathrm{id}\otimes v} X \times Y\Big).$$

Since we require that the map v is pure, this composite is also pure, and can thus be described as $\mathsf{emb}(\mathsf{assign}(v))\colon X \to \mathcal{D}_{\leq 1}(X \times Y)$.

We can now list the basic computations rules, including compositionality in the first point.

▶ **Lemma 4.** *The 'run' operation $- \Downarrow \omega$ from Definition 3 satisfies, for appropriately typed maps and predicates:*

1. $(f \,;\, g) \Downarrow \omega = g \Downarrow (f \Downarrow \omega)$, *i.e.* $\Big(\mathcal{D}(X) \xrightarrow{(f;g)\Downarrow -} \mathcal{D}(Z)\Big) = \Big(\mathcal{D}(X) \xrightarrow{f\Downarrow -} \mathcal{D}(Y) \xrightarrow{g\Downarrow -} \mathcal{D}(X)\Big)$
2. $\mathsf{asrt}_p \Downarrow \omega = \omega|_p$
3. $\mathsf{emb}(h) \Downarrow \omega = h_*(\omega)$, *where h is a 'pure' map in $\mathcal{K}\ell(\mathcal{D})$*
4. $(\mathsf{if}\ p\ \mathsf{then}\ f\ \mathsf{else}\ g) \Downarrow \omega = \frac{r}{r+s} \cdot \big(f \Downarrow \omega|_p\big) + \frac{s}{r+s} \cdot \big(g \Downarrow \omega|_{p^\perp}\big)$, *for* $r = \|f_*(p \cdot \omega)\|$ *and* $s = \|g_*(p^\perp \cdot \omega)\|$.

Proof. We prove the first two equations.

$$\begin{aligned}
\big(g \Downarrow (f \Downarrow \omega)\big)(z) &\overset{(8)}{=} \frac{\sum_y g(y)(z) \cdot (f \Downarrow \omega)(y)}{\sum_{y,z} g(y)(z) \cdot (f \Downarrow \omega)(y)} \\
&= \frac{\sum_y g(y)(z) \cdot \frac{f_*(\omega)(y)}{\|f_*(\omega)\|}}{\sum_{y,z} g(y)(z) \cdot \frac{f_*(\omega)(y)}{\|f_*(\omega)\|}} \\
&= \frac{\sum_{x,y} g(y)(z) \cdot f(x)(y) \cdot \omega(x)}{\sum_{x,y,z} g(y)(z) \cdot f(x)(y) \cdot \omega(x)} \\
&= \frac{\sum_x (f \,;\, g)(x)(z) \cdot \omega(x)}{\sum_{x,z} (f \,;\, g)(x)(z) \cdot \omega(x)} \;=\; \big((f \,;\, g) \Downarrow \omega\big)(z).
\end{aligned}$$

In the same way:

$$\big(\mathsf{asrt}_p \Downarrow \omega\big)(x) \overset{(8)}{=} \frac{\sum_{x'} \mathsf{asrt}_p(x')(x) \cdot \omega(x')}{\sum_{x,x'} \mathsf{asrt}_p(x')(x) \cdot \omega(x')} = \frac{p(x) \cdot \omega(x)}{\sum_x p(x) \cdot \omega(x)} \overset{(2)}{=} \omega|_p(x). \qquad ◀$$

The point of Lemma 4 is that the normalisation effect of the run operation is pushed through the whole program. This provides a formal justification of the fact that normalisation can be postponed to the end of a program. We briefly illustrate more concretely how this works.

▶ **Example 5.** Consider the program fragment `observe p; h; observe q`, represented categorically as:

$$c = \mathsf{asrt}_p \,;\, \mathsf{emb}(h) \,;\, \mathsf{asrt}_q, \qquad \text{where } h \text{ is pure.}$$

Running c in a proper state ω gives by Lemma 4:

$$\begin{aligned} c \Downarrow \omega \;=\; (\mathsf{asrt}_p \,;\, \mathsf{emb}(h) \,;\, \mathsf{asrt}_q) \Downarrow \omega &= (\mathsf{emb}(h) \,;\, \mathsf{asrt}_q) \Downarrow (\mathsf{asrt}_p \Downarrow \omega) \\ &= (\mathsf{emb}(h) \,;\, \mathsf{asrt}_q) \Downarrow \omega|_p \\ &= \mathsf{asrt}_q \Downarrow (\mathsf{emb}(h) \Downarrow \omega|_p) \\ &= \mathsf{asrt}_q \Downarrow h_*(\omega|_p) \\ &= h_*(\omega|_p)|_q. \end{aligned}$$

Similarly, for a program with parallelism:

$$d \;=\; (\mathsf{asrt}_p \otimes \mathrm{id}) \,;\, \mathsf{asrt}_q \,;\, (\mathrm{id} \otimes \mathsf{emb}(h)),$$

we get by using the assert-equations (5):

$$\begin{aligned} d \Downarrow \omega &= \big((\mathsf{asrt}_p \otimes \mathrm{id}) \,;\, \mathsf{asrt}_q \,;\, (\mathrm{id} \otimes \mathsf{emb}(h))\big) \Downarrow \omega \\ &= \big((\mathsf{asrt}_p \otimes \mathsf{asrt}_{\mathbf{1}}) \,;\, \mathsf{asrt}_q \,;\, (\mathsf{emb}(\mathrm{id}) \otimes \mathsf{emb}(h))\big) \Downarrow \omega \\ &= \big(\mathsf{asrt}_{p\otimes\mathbf{1}} \,;\, \mathsf{asrt}_q \,;\, \mathsf{emb}(\mathrm{id} \otimes h)\big) \Downarrow \omega \\ &= (\mathrm{id} \otimes h)_*\big((\omega|_{p\otimes\mathbf{1}})|_q\big) \\ &= (\mathrm{id} \otimes h)_*\big(\omega|_{(p\otimes\mathbf{1})\&q}\big). \end{aligned}$$

5 The monoid interpretation

This section describes an alternative semantics for probabilistic programs in the Kleisli category $\mathcal{K}\ell_{\mathcal{D}}(2 \times (-))$ on $\mathcal{K}\ell(\mathcal{D})$. A program is this a map of the form $X \to \mathcal{D}(2 \times Y)$, producing proper distributions.

▶ **Definition 6.** For a Kleisli map $c \colon X \to \mathcal{D}(2 \times Y)$, seen as a program, and an (initial) state $\omega \in \mathcal{D}(X)$ we define:

$$c \Downarrow \omega \;:=\; \mathsf{M}_2(c_*(\omega)|_{\mathrm{yes}\otimes\mathbf{1}}) \;\in\; \mathcal{D}(Y).$$

This 'run' starts from the transformed state $c_*(\omega) \in \mathcal{D}(2 \times Y)$. It is conditioned with the weakened predicate $\mathrm{yes} \otimes \mathbf{1} \in [0,1]^{2\times Y}$, where we write $\mathrm{yes} \in [0,1]^2$ for the predicate given by $\mathrm{yes} \mapsto 1$ and $\mathrm{no} \mapsto 0$. Subsequently, the second marginal yields a proper distribution on Y. Here we use what is called 'crossover' influence, where conditioning in one component of a product has an effect in the other component.

More concretely, the distribution $c \Downarrow \omega$ can be computed as:

$$\begin{aligned} (c \Downarrow \omega)(y) \;=\; \mathsf{M}_2(c_*(\omega)|_{\mathrm{yes}\otimes\mathbf{1}})(y) &= \textstyle\sum_{b\in 2} (c_*(\omega)|_{\mathrm{yes}\otimes\mathbf{1}})(b,y) \\ &= \textstyle\sum_{b\in 2} \dfrac{c_*(\omega)(b,y)\cdot(\mathrm{yes}\otimes\mathbf{1})(b,y)}{c_*(\omega) \models \mathrm{yes}\otimes\mathbf{1}} \\ &= \dfrac{c_*(\omega)(\mathrm{yes},y)}{\sum_{y,b} c_*(\omega)(b,y)\cdot(\mathrm{yes}\otimes\mathbf{1})(b,y)} \\ &= \dfrac{\sum_x c(x)(\mathrm{yes},y)\cdot\omega(x)}{\sum_{x,y} c(x)(\mathrm{yes},y)\cdot\omega(x)}. \end{aligned} \qquad (9)$$

The same basic programs as in the previous section can be interpreted as Kleisli maps in $\mathcal{K}\ell_{\mathcal{D}}(2 \times (-))$. This is briefly described below.

1. Observation of a predicate p on X is now translated into an instrument map $\mathsf{instr}_p \colon X \to \mathcal{D}(2 \times X)$.
2. The statement $\mathsf{if}\ p\ \mathsf{then}\ f\ \mathsf{else}\ g \colon X \to \mathcal{D}_{\leq 1}(Y)$, for $p \in [0,1]^X$ and $f, g \colon X \to \mathcal{D}(2 \times Y)$, is interpreted basically in the same way as in the one-deficit case, namely in $\mathcal{K}\ell(\mathcal{D})$ as:
$$\mathsf{if}\ p\ \mathsf{then}\ f\ \mathsf{else}\ g \;\coloneqq\; \Big(X \xrightarrow{\mathsf{instr}_p} 2 \times X \cong X + X \xrightarrow{[f,g]} 2 \times Y\Big).$$
3. The embedding statement $\mathsf{emb}(h) \colon X \to \mathcal{D}(2 \times Y)$, for a 'pure' map $h \colon X \to \mathcal{D}(Y)$ is obtained via post-composition with the unit $Y \to 2 \times Y$, given by $y \mapsto 1|\text{yes}, y\rangle$ of the monad $2 \times (-)$ on $\mathcal{K}\ell(\mathcal{D})$.

   We can prove the analogue of Lemma 4.

▶ **Lemma 7.** *The 'run' operation $- \Downarrow \omega$ from Definition 6 satisfies:*
1. $(f \,;\, g) \Downarrow \omega = g \Downarrow (f \Downarrow \omega)$
2. $\mathsf{instr}_p \Downarrow \omega = \omega|_p$
3. $\mathsf{emb}(h) \Downarrow \omega = h_*(\omega)$*, where h is 'pure'*
4. $(\mathsf{if}\ p\ \mathsf{then}\ f\ \mathsf{else}\ g) \Downarrow \omega = \frac{r}{r+s} \cdot \big(f \Downarrow \omega|_p\big) + \frac{s}{r+s} \cdot \big(g \Downarrow \omega|_{p^\perp}\big)$*, for* $r = \sum_y f_*(p \cdot \omega)(\text{yes}, y)$ *and* $s = \sum_y g_*(p^\perp \cdot \omega)(\text{yes}, y)$.

Proof. We do the first two cases:
$$\begin{aligned}
\big(g \Downarrow (f \Downarrow \omega)\big)(z) &\overset{(9)}{=} \frac{\sum_y g(y)(\text{yes}, z) \cdot (f \Downarrow \omega)(y)}{\sum_{y,z} g(y)(\text{yes}, z) \cdot (f \Downarrow \omega)(y)} \\
&= \frac{\sum_y g(y)(\text{yes}, z) \cdot \frac{\sum_x f(x)(\text{yes},y) \cdot \omega(x)}{f_*(\omega) \models \text{yes} \otimes \mathbf{1}}}{\sum_y g(y,z)(\text{yes}, z) \cdot \frac{\sum_x f(x)(\text{yes},y) \cdot \omega(x)}{f_*(\omega) \models \text{yes} \otimes \mathbf{1}}} \\
&= \frac{\sum_{x,y} g(y)(\text{yes}, z) \cdot f(x)(\text{yes}, y) \cdot \omega(x)}{\sum_{x,y,z} g(y)(\text{yes}, z) \cdot f(x)(\text{yes}, y) \cdot \omega(x)} \\
&\overset{(*)}{=} \frac{\sum_x (f \,;\, g)(x)(\text{yes}, z) \cdot \omega(x)}{\sum_{x,z} (f \,;\, g)(x)(\text{yes}, z) \cdot \omega(x)} \overset{(9)}{=} \big((f \,;\, g) \Downarrow \omega\big)(z).
\end{aligned}$$
In the marked equation $\overset{(*)}{=}$ we make crucial use of the fact that the Boolean monoid 2 satisfies $b_1 \wedge b_2 = \text{yes}$ implies $b_1 = b_2 = \text{yes}$. Next:
$$\big(\mathsf{instr}_p \Downarrow \omega\big)(x) \overset{(9)}{=} \frac{\sum_{x'} \mathsf{instr}_p(x')(\text{yes}, x) \cdot \omega(x')}{\sum_{x,x'} \mathsf{instr}_p(x')(\text{yes}, x) \cdot \omega(x')} = \frac{p(x) \cdot \omega(x)}{\sum_x p(x) \cdot \omega(x)} = \omega|_p(x).$$ ◀

5.1 Relating the one-deficit and the monoid interpretations

The following result relates our two program interpretations.

▶ Proposition 8. There is a functor $\mathcal{Y}$, for 'yes', between the categories for the monoid and the one-deficit interpretations, in a commuting diagram:
$$\begin{tikzcd}
\mathcal{K}\ell_{\mathcal{D}}(2 \times (-)) \arrow[rr, "\mathcal{Y}"] & & \mathcal{K}\ell_{\mathcal{D}}((-) + 1) \\
& \mathcal{K}\ell(\mathcal{D}) \arrow[ul] \arrow[ur] &
\end{tikzcd} \tag{10}$$
On objects it is the identity: $\mathcal{Y}(X) = X$, and on morphisms it is:
$$\Big(X \xrightarrow{f} 2 \times Y\Big) \overset{\mathcal{Y}}{\longmapsto} \Big(X \xrightarrow{f} 2 \times Y \cong Y + Y \xrightarrow{\mathrm{id}+!} Y + 1\Big).$$

More explicitly, $\mathcal{Y}(f)(x)(y) = f(x)(\text{yes}, y)$.

This $\mathcal{Y}$ is a strictly monoidal functor, thus preserving sequential and parallel composition. The commutativity of (10) means that $\mathcal{Y}$ preserves embed: $\mathcal{Y}(\mathsf{emb}(h)) = \mathsf{emb}(h)$. Moreover it preserves the basic program constructs for observe and if-then-else:

$$\mathcal{Y}(\mathsf{instr}_p) = \mathsf{asrt}_p \qquad \mathcal{Y}(\mathsf{if}\ p\ \mathsf{then}\ f\ \mathsf{else}\ g) = \mathsf{if}\ p\ \mathsf{then}\ \mathcal{Y}(f)\ \mathsf{else}\ \mathcal{Y}(g).$$

In addition, 'run' is preserved: $\mathcal{Y}(c) \Downarrow \omega = c \Downarrow \omega$, for each $c\colon X \to \mathcal{D}(2 \times Y)$ and $\omega \in \mathcal{D}(X)$.

With this result, Lemma 7 follows in fact from Lemma 4.

Proof. A basic observation that simplifies the proof at an abstract level is that the composite:

$$2 \times Y \cong Y + Y \xrightarrow{\mathrm{id}+!} Y + 1 \quad \text{in } \mathcal{K}\ell(\mathcal{D}) \qquad \text{is} \qquad 2 \times Y \xrightarrow{\text{yes}\otimes\mathrm{id}} 1 \times Y \cong Y \quad \text{in } \mathcal{K}\ell_{\mathcal{D}}((-)+1)$$

where yes: $2 \to 1$ is the 'yes' predicate, as a map in $\mathcal{K}\ell_{\mathcal{D}}((-)+1)$. Let us write $\star$ and $\odot$ for the compositions in $\mathcal{K}\ell_{\mathcal{D}}(2\times(-))$ and $\mathcal{K}\ell_{\mathcal{D}}((-)+1)$, respectively, and $\langle - \rangle\colon \mathcal{K}\ell(\mathcal{D}) \to \mathcal{K}\ell_{\mathcal{D}}((-)+1)$ for the embedding functor. We then have $\mathcal{Y}(f) = (\text{yes}\otimes\mathrm{id}_Y)\circ f = (\text{yes}\otimes\mathrm{id}_Y)\odot\langle f\rangle$, suppressing the coherence isomorphism, and for $f\colon X \to 2 \times Y$ and $g\colon Y \to 2 \times Z$,

$$\begin{aligned}
\mathcal{Y}(g \star f) &= (\text{yes} \otimes \mathrm{id}_Y) \odot \langle g \star f\rangle = (\text{yes} \otimes \mathrm{id}_Y) \odot \langle(\wedge \otimes \mathrm{id}_Z) \circ (\mathrm{id}_2 \otimes g) \circ f)\rangle \\
&= (\text{yes} \otimes \mathrm{id}_Y) \odot (\langle\wedge\rangle \otimes \mathrm{id}_Z) \odot (\mathrm{id}_2 \otimes \langle g\rangle) \odot \langle f\rangle \\
&= (\text{yes} \otimes \text{yes} \otimes \mathrm{id}_Z) \odot (\mathrm{id}_2 \otimes \langle g\rangle) \odot \langle f\rangle \\
&= (\text{yes} \otimes \mathrm{id}_Z) \odot \langle g\rangle \odot (\text{yes} \otimes \mathrm{id}_Y) \odot \langle f\rangle \\
&= \mathcal{Y}(g) \odot \mathcal{Y}(f),
\end{aligned}$$

where we use $\text{yes} \odot \langle\wedge\rangle = \text{yes} \otimes \text{yes}$ for the conjunction map $\wedge\colon 2 \times 2 \to 2$. Similarly we prove that $\mathcal{Y}(\mathrm{id}) = \mathrm{id}$ and $\mathcal{Y}(f \otimes g) = \mathcal{Y}(f) \otimes \mathcal{Y}(g)$, and that diagram (10) commutes.

It is easy to see that the program constructs for observe and if-then-else are preserved by the functor $\mathcal{Y}$, and so we concentrate on preservation of run:

$$(\mathcal{Y}(c) \Downarrow \omega)(y) \overset{(8)}{=} \frac{\sum_x \mathcal{Y}(c)(x)(y) \cdot \omega(x)}{\sum_{x,y} \mathcal{Y}(c)(x)(y) \cdot \omega(x)} = \frac{\sum_x c(x)(\text{yes}, y) \cdot \omega(x)}{\sum_{x,y} c(x)(\text{yes}, y) \cdot \omega(x)} \overset{(9)}{=} (c \Downarrow \omega)(y) \qquad \blacktriangleleft$$

6 Fish-in-a-pond example

Both the one-deficit and the monoid semantics that we described in the previous two sections have been elaborated in an experimental implementation in Python, via the EfProb library[2] for probabilistic computation. Below we give an impression of how this works. We present the example first via informal pseudo code, as in Example 1, and then discuss the semantics. We shall describe the monoid form below. The one-deficit implementation works similarly. Interestingly, the one-deficit implementation runs considerably faster since it does not double the state space via multiplication with 2. The translation of the code to EfProb is done by hand. In follow-up work we will do this more systematically. We emphasise that our implementation precisely calculates distributions, in contrast to sample based computations that approximate the result.

[2] Available via `efprob.cs.ru.nl`; the whole representation of the examples will be made available there.

Imagine we are looking at a pond and we wish to learn the number of fish in it. We catch twenty fish, 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? This technique is used to count populations and is known as mark-and-recapture.

The problem can be modelled by the following probabilistic program.

```
x <- uniform(fish_dom);
y <- binomial(20, 20 / x);
observe (y = 5);                                    (11)
discard y
```

Here `fish_dom` is a set of possible numbers of fish that we consider, say `fish_dom` $= \{20, 30, 40, \ldots, 250\}$ with units of 10 fish. Then `uniform(fish_dom)` is the uniform distribution on this set. The variable x takes a value sampled from the distribution. We use a uniform distribution because we assume no prior knowledge about the number of fish in the pond. Next, `binomial(20, 20 / x)` is the binomial distribution with parameters $n = 20$ and probability $p = 20/x$. It can be described explicitly as $\sum_{k<21} p^k(1-p)^{n-k}|k\rangle$ in $\mathcal{D}(\{0, 1, \ldots, 20\})$. It gives for each k the probability of getting k marked fish among the 20 that we catch (the second time). We use a binomial distribution here, assuming for simplicity that we catch these 20 fish one by one, check if they are marked, and then throw them back. Because the variable y is discarded, this program returns only x and thus is interpreted as a function $1 \to \mathcal{D}(2 \times \texttt{fish\_dom})$.

We compute the interpretation and its 'run' in a systematic way using Python with the EfProb library. The program (11) is translated (manually) into the following python code.

```python
seq(assign(uniform_state(fish_dom)),
    assign(chan_fromklmap(lambda x: binomial(20, 20 / x),
                          fish_dom, range(21))),
    observe(truth(fish_dom) @ point_pred(5, range(21))),
    idn(fish_dom) @ discard(range(21)))
```

Here `seq(f, g, ...)` is sequential composition $f; g; \cdots$, and `f @ g` is parallel composition $f \otimes g$. The code is evaluated into a 'channel' in the EfProb library, of type $1 \to 2 \times \texttt{fish\_dom}$. We can then compute its 'run' by conditioning and marginalisation from Definition 6, which are basic operations in EfProb. The final state on `fish_dom` is plotted in Figure 1. The expected value of this state, *i.e.* the number of fish in the pond, is 112.

7 Combining the partial and the monoid interpretations

So far we have seen two interpretations for conditioning, namely the one-deficit interpretation using the lift monad $(-) + 1$ and asserts, and the monoid interpretation using the monoid monad $2 \times (-)$ and instruments. In this final section we briefly look at the options for extending our language with partiality, via an abort statement that never terminates. For this purpose we use the lift monad $(-) + 1$ for its original purpose, namely partiality.

We shall look at the following two combinations of monoid and lift:

1. "inner monoid", of the form $(2 \times -) + 1$;
2. "outer monoid", of the form $2 \times (- + 1)$.

We start with the first option, and illustrate why it does not work. For the second approach we show that the examples work well, and that the analogue of Lemma 7 holds. We have no implementation of the outer monoid approach since EfProb does not support lift (or coproducts).

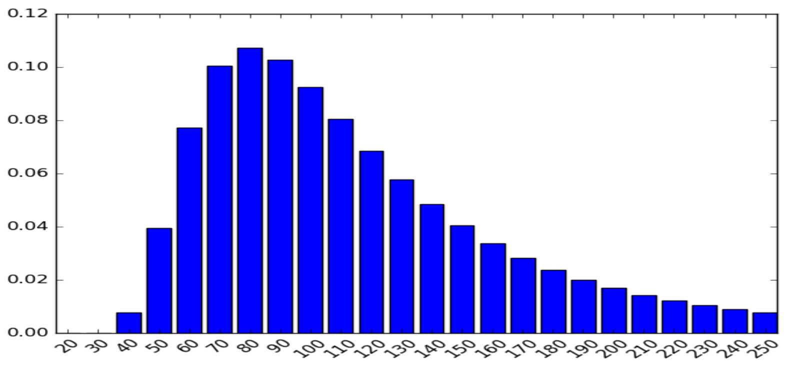

Figure 1 The state after the program runs

7.1 The inner monoid approach

We consider the combination $(2\times-)+1$. It will be used in the form of the monad $\mathcal{D}_{\leq 1}(2\times(-))$ on **Set**. The interpretation of our internal language, with observe (via instruments), embed, if-then-else, is basically as described in Section 5, but now we have an additional statement $\mathsf{abort}\colon X \to \mathcal{D}_{\leq 1}(2\times Y)$, given by $\mathsf{abort}(x)(b,y) = 0$ for all $x\in X, b\in 2, y\in Y$. Notice that $\mathsf{abort} \mathbin{;} f = \mathsf{abort} = g \mathbin{;} \mathsf{abort}$ and also $\mathsf{abort}\otimes f = \mathsf{abort} = g\otimes\mathsf{abort}$.

In the current combined setting we have to define a 'run', just like in Definitions 3 and 6. For a Kleisli map $c\colon X\to\mathcal{D}_{\leq 1}(2\times Y)$, seen as a program, and an (initial) state $\omega\in\mathcal{D}(X)$ we define, using 'liberal' conditioning $|^{\ell}$ from Subsection 3.1:

$$c \Downarrow \omega \;:=\; \mathsf{M}_2\big(c_*(\omega)|^{\ell}_{\mathrm{yes}\otimes\mathbf{1}}\big) \;\in\; \mathcal{D}_{\leq 1}(Y). \tag{12}$$

Following Equations (8) and (9) it is not hard to see that:

$$\big(c\Downarrow\omega\big)(y) \;=\; \frac{\sum_x c(x)(\mathrm{yes},y)\cdot\omega(x)}{1-\sum_{x,y} c(x)(\mathrm{no},y)\cdot\omega(x)}. \tag{13}$$

We shall illustrate this inner monoid approach in two examples, one where it works well, and one where it does not.

▶ **Example 9.** Consider the following simple probabilistic program with non-termination in a convex sum, taken from [7].

```
{abort} $+_{1/2}$ {{x := 0} $+_{1/2}$ {x := 1};
                  {y := 0} $+_{1/2}$ {y := 1};
                  observe (x = 0 $\vee$ y = 0) }
```

The question is: what is the probability that `y` is zero in the final state?

We simply use the set $N=\{0,1\}$ as domain of the variables `x`, `y`, since the program involves only the two numbers 0 and 1. The program is interpreted as a Kleisli map $1\to N\times N$ for the monad $\mathcal{D}_{\leq 1}(2\times(-))$, *i.e.* as a function $c\colon 1\to\mathcal{D}_{\leq 1}(2\times N\times N)$. Identified with a subdistribution, this c is given as:

$$\tfrac{1}{8}|\mathrm{yes},0,0\rangle + \tfrac{1}{8}|\mathrm{yes},0,1\rangle + \tfrac{1}{8}|\mathrm{yes},1,0\rangle + \tfrac{1}{8}|\mathrm{no},1,1\rangle \;\in\mathcal{D}_{\leq 1}(2\times N\times N)$$

By applying 'run' from (12), formally by computing $c\Downarrow\omega$ with trivial initial state $\omega = 1|*\rangle$, the unique state on 1, we obtain the state:

$$\tfrac{1}{7}|0,0\rangle + \tfrac{1}{7}|0,1\rangle + \tfrac{1}{7}|1,0\rangle \tag{14}$$

In [7] the so-called cwp-semantics yields that the probability of $y = 0$ after execution of the program is $\frac{2}{7}$. Indeed we obtain $\frac{2}{7}|0\rangle + \frac{1}{7}|1\rangle$ by marginalising the first component of (14).

▶ **Example 10.** In the next example we use abort inside an if-then-else:

```
observe (x = 0);
if (x = 0) { skip } else { abort }
```

Imagine that this program is run with initial state $\omega = \frac{1}{2}|0\rangle + \frac{1}{2}|1\rangle$. What should be the outcome? Surely the state $1|0\rangle$.

Suppose that the domain of the variable x is the set $N = \{0, 1\}$. The interpretation of the first and second line will be described as channels $f_1, f_2 \colon N \to \mathcal{D}((2 \times N) + 1)$. The first line gives $f_1(0) = 1|\text{yes}, 0\rangle$ and $f_2(1) = 1|\text{no}, 1\rangle$. The second line is interpreted as $f_2(0) = 1|\text{yes}, 0\rangle$ and $f_2(1) = 1|*\rangle$. The whole program is then interpreted as the Kleisli composition f, where:

$$f(x) \;=\; (f_2 \circ f_1)(x) \;=\; \begin{cases} 1|\text{yes}, 0\rangle & \text{if } x = 0 \\ 1|*\rangle & \text{if } x = 1. \end{cases}$$

The problem of this interpretation is that one loses the information that the diverging run is blocked by the observe command. Indeed, if we compute the run (12) with initial state $\omega = \frac{1}{2}|0\rangle + \frac{1}{2}|1\rangle$, we obtain $f \Downarrow \omega = \frac{1}{2}|0\rangle + \frac{1}{2}|*\rangle$, and not $1|0\rangle$ as it should be. Hence the inner monoid semantics is wrong.

There is another aspect that is problematic with the inner monoid semantics, namely the equation $g \Downarrow (f \Downarrow \omega) = (f \,;\, g) \Downarrow \omega$, see Lemmas 4 and 7, fails. The channels f_1 and f_2 in Example 10 give a counterexample.

7.2 The outer monoid approach

We check that the outer monoid approach does properly handle the programs from Examples 9 and 10. The statement $\mathsf{abort} \colon X \to \mathcal{D}(2 \times (X + 1))$ is interpreted as $\mathsf{abort}(x) = 1|\text{yes}, *\rangle$, where $*$ is the sole element of 1.

For a function $c \colon X \to \mathcal{D}(2 \times (Y + 1))$ and a state $\omega \in \mathcal{D}(X)$ we now define:

$$c \Downarrow \omega \;:=\; \mathsf{M}_2\big(c_*(\omega)|_{\text{yes}\otimes\mathbf{1}}\big) \;\in\; \mathcal{D}(Y + 1). \tag{15}$$

▶ **Example 11.** Under the outer interpretation the program from Example 9 yields a state:

$$\tfrac{1}{2}|\text{yes}, *\rangle + \tfrac{1}{8}|\text{yes}, 0, 0\rangle + \tfrac{1}{8}|\text{yes}, 0, 1\rangle + \tfrac{1}{8}|\text{yes}, 1, 0\rangle + \tfrac{1}{8}|\text{no}, 1, 1\rangle.$$

Following the description of 'run' in (15) we normalise the 'yes' occurrences and take the second marginal, resulting in:

$$\begin{aligned} \mathsf{M}_2\Big(\tfrac{1/2}{7/8}|*\rangle + \tfrac{1/8}{7/8}|0,0\rangle + \tfrac{1/8}{7/8}|,0,1\rangle + \tfrac{1/8}{7/8}|1,0\rangle\Big) \;&=\; \mathsf{M}_2\Big(\tfrac{4}{7}|*\rangle + \tfrac{1}{7}|0,0\rangle + \tfrac{1}{7}|0,1\rangle + \tfrac{1}{7}|1,0\rangle\Big) \\ &=\; \tfrac{4}{7}|*\rangle + \tfrac{2}{7}|0\rangle + \tfrac{1}{7}|1\rangle. \end{aligned}$$

▶ **Example 12.** Let's now write $g_1, g_2 \colon N \to \mathcal{D}(2 \times (N+1))$ for the interpretations of the first and second line of the program in Example 10. Then: $g_1(0) = 1|\text{yes}, 0\rangle$, $g_1(1) = 1|\text{no}, 1\rangle$ and $g_2(0) = 1|\text{yes}, 0\rangle$, $g_2(1) = 1|\text{yes}, *\rangle$. The composite $g = g_2 \circ g_1$ then satisfies $g(0) = 1|\text{yes}, 0\rangle$ and $g(1) = 1|\text{no}, *\rangle$. Applying it to the initial state $\omega = \frac{1}{2}|0\rangle + \frac{1}{2}|1\rangle$ we get the desired outcome, since, according to (15),

$$g \Downarrow \omega \;=\; \mathsf{M}_2\Big(\big(\tfrac{1}{2}|\text{yes}, 0\rangle + \tfrac{1}{2}|\text{no}, *\rangle\big)|_{\text{yes}\otimes\mathbf{1}}\Big) \;=\; \mathsf{M}_2\Big(\tfrac{1/2}{1/2}|\text{yes}, 0\rangle\Big) \;=\; 1|0\rangle.$$

In this outer monoid approach the 'run' operation satisfies the 'computation rules'.

▶ **Lemma 13.** *The analogue of Lemma 7 holds for the outer monoid approach.* ◀

The details will appear in follow-up work.

8 Conclusions

This papers describes several ways of interpreting probabilistic programs with conditioning in Kleisli categories. The two main approaches are the 'one-deficit' approach using the lift monad $(-) + 1$, and the 'monoid approach' using the monad $2 \times (-)$. These approaches are described abstractly in categorical terms, but ultimately also produce running code in Python. Finally, the subtleties of including non-termination are briefly discussed.

References

1 J. Borgström, A.D. Gordon, M. Greenberg, J. Margetson, and J. Van Gael. Measure transformer semantics for Bayesian machine learning. *Logical Methods in Comp. Sci.*, 9(3):1–39, 2013. `doi:10.2168/LMCS-9(3:11)2013`.

2 K. Cho, B. Jacobs, A. Westerbaan, and B. Westerbaan. An introduction to effectus theory. Preprint, 2015. arXiv:1512.05813 [cs.LO].

3 A. Gordon, T. Henzinger, A. Nori, and S. Rajamani. Probabilistic programming. In *Future of Software Engineering*, pages 167–181. ACM, 2014. `doi:10.1145/2593882.2593900`.

4 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`.

5 B. Jacobs. *Introduction to Coalgebra: Towards Mathematics of States and Observation.* Cambridge Univ. Press, 2016.

6 B. Jacobs. From probability monads to commutative effectuses. *Journ. of Logical and Algebraic Methods in Programming*, 2017, to appear.

7 N. Jansen, B. Lucien Kaminski, J.-P. Katoen, F. Olmedo, F. Gretz, and A. McIver. Conditioning in probabilistic programming. In *MFPS XXXI*, volume 319 of *ENTCS*, pages 199–216. Elsevier, 2015. `doi:10.1016/j.entcs.2015.12.013`.

8 B. Lucien Kaminski, J.-P. Katoen, C. Matheja, and F. Olmedo. Weakest precondition reasoning for expected run-times of probabilistic programs. In *ESOP 2016*, volume 9632 of *LNCS*, pages 364–389. Springer, 2016. `doi:10.1007/978-3-662-49498-1_15`.

9 J.-P. Katoen, F. Gretz, N. Jansen, B. Lucien Kaminski, and F. Olmedo. Understanding probabilistic programs. In *Correct System Design*, volume 9360 of *LNCS*, pages 15–32. Springer, 2015. `doi:10.1007/978-3-319-23506-6_4`.

10 F. Olmedo, B. Lucien Kaminski, J.-P. Katoen, and C. Matheja. Reasoning about recursive probabilistic programs. In *LICS 2016*, pages 672–681. ACM, 2016. `doi:10.1145/2933575.2935317`.

11 S. Staton. Commutative semantics for probabilistic programming. To appear in *ESOP 2017*.

12 S. Staton, H. Yang, F. Wood, C. Heunen, and O. Kammar. Semantics for probabilistic programming: Higher-order functions, continuous distributions, and soft constraints. In *LICS 2016*, pages 525–534. ACM, 2016. `doi:10.1145/2933575.2935313`.