Simplifying Probabilistic Programs Using Computer Algebra*

Jacques Carette[1] and Chung-chieh Shan[2]

[1] McMaster University carette@mcmaster.ca
[2] Indiana University ccshan@indiana.edu

Abstract. We transform probabilistic programs to run more efficiently and read more easily, by composing three semantics-preserving transformations: (1) apply the denotational semantics; (2) improve the resulting integral; then (3) invert the denotational semantics. Whereas step 1 is a straightforward transformation from monadic to continuation-passing style, the rest builds on computer algebra: step 2 reorders and performs integrals, and step 3 represents density functions as differential operators.

Keywords: probabilistic programming, computer algebra, integrating continuous distributions

1 Introduction

The success of machine learning has made it clear that computing with probability distributions is very useful. Given a distribution, we might want a simpler representation of it or to generate random samples from it. These two computations go hand in hand: If we are lucky, we can find a simple representation, maybe even an exact one, that renders all further calculation trivial. If not, we would likely want to generate random samples, a process that can be made dramatically more efficient and accurate by any simplification we manage to perform.

1.1 Contributions

We introduce a way to simplify probabilistic programs: starting with a monadic representation of a distribution (Section 2), we transform it to an integral (Section 4), improve the integral by controlled use of computer algebra (Section 6), then transform back (Section 5). Whereas the denotational semantics that maps probabilistic programs to integrals is straightforward and well known (Section 3), we put it to work and show how to turn integrals into simpler programs. In

* Thanks to Mike Kucera, Praveen Narayanan, Natalie Perna, and Robert Zinkov for developing the Hakaru probabilistic programming system, the home of our research. This research was supported by DARPA grant FA8750-14-2-0007, NSF grant CNS-0723054, Lilly Endowment, Inc. (through its support for the Indiana University Pervasive Technology Institute), and the Indiana METACyt Initiative. The Indiana METACyt Initiative at IU is also supported in part by Lilly Endowment, Inc.

particular, we identify necessary transformations on mathematical and integral expressions that direct an existing computer algebra system (CAS) to simplify the kind of multidimensional integrals that probabilistic programs tend to mean. We also apply computer algebra to convert integrals back to programs robustly. The CAS we use is Maple, but other systems such as Mathematica are similar.

1.2 Three examples

We showcase our approach using three natural (if small) examples.

A discrete distribution Start with the following sampling procedure: choose two real numbers x, y independently from the uniform distribution between 0 and 1, then return whether $x < y$. The corresponding term in our language

$$\mathsf{Bind}(\mathsf{Uniform}(0,1), x, \mathsf{Bind}(\mathsf{Uniform}(0,1), y, \mathsf{If}(x < y, \mathsf{Ret}(\mathsf{true}), \mathsf{Ret}(\mathsf{false})))) \quad (1)$$

denotes a measure over Booleans ($\mathbb{B}$). In other words, we can integrate with respect to this distribution any function $h : \mathbb{B} \to \mathbb{R}^+$. The result is the number

$$\int_0^1 \int_0^1 \left(\begin{cases} h(\mathsf{true}) & \text{if } x < y \\ h(\mathsf{false}) & \text{otherwise} \end{cases} \right) dy\, dx. \quad (2)$$

Out of the box, a CAS like Maple can simplify this integral expression to

$$\frac{1}{2} \cdot h(\mathsf{true}) + \frac{1}{2} \cdot h(\mathsf{false}). \quad (3)$$

From this integral, we read off the simpler term in our language

$$\mathsf{Msum}\big(\mathsf{Weight}(1/2, \mathsf{Ret}(\mathsf{true})), \mathsf{Weight}(1/2, \mathsf{Ret}(\mathsf{false}))\big). \quad (4)$$

This term denotes the same measure over Booleans as (1), but it is more efficient as a sampling procedure: instead of drawing two numbers from a uniform distribution, just flip a single fair coin and return either true or false accordingly. This term is also easier for humans to understand. Indeed, this representation amounts to a compact table that lists the outcomes (true and false) alongside their exact probabilities (1/2 each). Nothing could be better.

A continuous distribution Let us take two steps on a one-dimensional random walk: first choose x from the Gaussian distribution with mean 0 and standard deviation 1, then choose y from the Gaussian distribution with mean x and standard deviation 1. If we only care about the final point y, then the corresponding term in our language

$$\mathsf{Bind}(\mathsf{Gaussian}(0,1), x, \mathsf{Bind}(\mathsf{Gaussian}(x,1), y, \mathsf{Ret}(y))) \quad (5)$$

denotes a measure over $\mathbb{R}$. In other words, we can integrate with respect to this distribution any function h from $\mathbb{R}$ to $\mathbb{R}^+$. The result is the number

$$\int_{-\infty}^{\infty} \frac{\exp\left(-\frac{x^2}{2}\right)}{\sqrt{2 \cdot \pi}} \cdot \int_{-\infty}^{\infty} \frac{\exp\left(-\frac{(y-x)^2}{2}\right)}{\sqrt{2 \cdot \pi}} \cdot h(y)\, dy\, dx. \quad (6)$$

With Maple's help, we can simplify this integral expression to

$$\int_{-\infty}^{\infty} \frac{\exp\left(-\frac{y^2}{4}\right)}{2 \cdot \sqrt{\pi}} \cdot h(y)\, dy. \tag{7}$$

From this, we read off the much simpler term in our language $\mathsf{Gaussian}(0, \sqrt{2})$, which expresses the fact that taking two steps on a random walk with standard deviation 1 is equivalent to taking one step on a random walk with standard deviation $\sqrt{2}$. This term denotes the same measure over $\mathbb{R}$ as (5), but it is more efficient as a sampling procedure: instead of drawing two numbers from Gaussian distributions, just draw one number from a third Gaussian. This term is also easier for humans to understand (for instance, easier to visualize as a bell curve).

A conditional distribution What if, instead of caring about the second step y and throwing away x, we *observe* y from an actual walk and want to use that information to *infer* the first step x? One way to express the observation is to replace the random choice of y in (5) by a *probability density* $\mathcal{D}$ (Table 1):

$$\mathsf{Bind}(\mathsf{Gaussian}(0, 1), x, \mathsf{Weight}(\mathcal{D}(\mathsf{Gaussian}(x, 1), y), \mathsf{Ret}(x))) \tag{8}$$

The integral of h with respect to this distribution is similar to (6),

$$\int_{-\infty}^{\infty} \frac{\exp\left(-\frac{x^2}{2}\right)}{\sqrt{2 \cdot \pi}} \cdot \frac{\exp\left(-\frac{(y-x)^2}{2}\right)}{\sqrt{2 \cdot \pi}} \cdot h(x)\, dx. \tag{9}$$

From this integral, we read off the term

$$\mathsf{Weight}\left(\frac{\exp\left(-\frac{y^2}{4}\right)}{2 \cdot \sqrt{\pi}}, \mathsf{Gaussian}\left(\frac{y}{2}, \frac{1}{\sqrt{2}}\right)\right) \tag{10}$$

for the *conditional* distribution of x *given* the observation y. This term tells us that this distribution is proportional to a Gaussian with mean $y/2$ and standard deviation $1/\sqrt{2}$. It denotes the same measure over $\mathbb{R}$ as (8), but it is more efficient as a sampling procedure because the weight $\exp(-y^2/4)/(2 \cdot \sqrt{\pi})$ does not vary from one sample of x to the next. This description of the distribution is also the basis of the famous *Kalman filter* (Maybeck 1979).

2 Adding measures to a mathematical language

Instead of building our own language from scratch, we start with a CAS whose language already expresses integral calculus. For example, Maple represents the integral (2) internally as the term

$$\mathsf{Int}\big(\mathsf{Int}\big(\mathsf{If}(x < y, h(\mathsf{true}), h(\mathsf{false})), y = 0..1\big), x = 0..1\big) \tag{11}$$

using the syntax constructors $\mathsf{Int}\ \mathsf{If}\ \mathsf{true}\ \mathsf{false} < = ..$ (the last three of which are written infix) and the variables x y and h.

$$\frac{a:\mathbb{R} \quad b:\mathbb{R}}{\mathsf{Uniform}(a,b):\mathbb{MR}} \qquad \frac{\mu:\mathbb{R} \quad \sigma:\mathbb{R}^+}{\mathsf{Gaussian}(\mu,\sigma):\mathbb{MR}} \qquad \frac{\mu:\mathbb{R} \quad \gamma:\mathbb{R}^+}{\mathsf{Cauchy}(\mu,\gamma):\mathbb{MR}}$$

$$\frac{\nu:\mathbb{R} \quad \mu:\mathbb{R} \quad \gamma:\mathbb{R}^+}{\mathsf{StudentT}(\nu,\mu,\gamma):\mathbb{MR}} \qquad \frac{\alpha:\mathbb{R}^+ \quad \beta:\mathbb{R}^+}{\mathsf{Beta}(\alpha,\beta):\mathbb{MR}^+} \qquad \frac{k:\mathbb{R}^+ \quad \theta:\mathbb{R}^+}{\mathsf{Gamma}(k,\theta):\mathbb{MR}^+}$$

$$\frac{e:A}{\mathsf{Ret}(e):\mathbb{M}A} \qquad \frac{m:\mathbb{M}A \quad \begin{array}{c}[x:A]\\ \vdots \\ m':\mathbb{M}B\end{array}}{\mathsf{Bind}(m,x,m'):\mathbb{M}B} \qquad \frac{\forall i \le n,\ m_i:\mathbb{M}A}{\mathsf{Msum}(m_1,\ldots,m_n):\mathbb{M}A}$$

$$\frac{e:\mathbb{R}^+ \quad m:\mathbb{M}A}{\mathsf{Weight}(e,m):\mathbb{M}A} \qquad \frac{e:\mathbb{B} \quad m:\mathbb{M}A \quad m':\mathbb{M}A}{\mathsf{If}(e,m,m'):\mathbb{M}A} \qquad \frac{\begin{array}{c}[\mathfrak{h}:A\to\mathbb{R}^+]\\ \vdots \\ e:\mathbb{R}^+\end{array}}{\mathsf{LO}(\mathfrak{h},e):\mathbb{M}A}$$

Fig. 1. Informal typing rules for how we represent measures

We add to this language a handful of constructors that amount to an abstract data type of measures. These constructors are summarized in Figure 1, using informal "typing rules" even though Maple is not statically typed. If A is a type, then $\mathbb{M}A$ is our informal type of measures over A. In particular, the top two rows of Figure 1 show several primitive measures of types $\mathbb{MR}$ and $\mathbb{MR}^+$, where $\mathbb{R}$ is the type of reals and $\mathbb{R}^+$ is the type of nonnegative reals (including $+\infty$).

Moving on in Figure 1, the constructors Ret and Bind represent the unit and bind operations of the measure monad. In $\mathsf{Bind}(m,x,m')$, the variable x takes scope over the measure term m'. Here we write x as a metavariable that ranges over variables in the syntax; whereas for terms we use a variety of metavariables, including $e,a,b,k,\alpha,\beta,\gamma,\mu,\nu,\sigma,\theta$, and especially m for a measure term, and g for an integral term.

The next two constructors express nonnegative linear combinations of measures. The term $\mathsf{Msum}(m_1,\ldots,m_n)$ represents the sum of n measures, so $\mathsf{Msum}()$ is the zero measure. The term $\mathsf{Weight}(e,m)$ represents multiplying a measure m by a nonnegative scalar factor e.

We use the conditional If in measure terms as well as in ordinary expressions denoting numbers. Thus $\mathsf{If}(x<y,\mathsf{Ret}(\mathsf{true}),\mathsf{Ret}(\mathsf{false}))$ is a measure term, whereas the expression $\mathsf{If}(x<y,h(\mathsf{true}),h(\mathsf{false}))$ denotes a number. In our Maple implementation, we actually handle the multiway conditional construct piecewise (Carette 2007), but we describe only if-then-else in this paper to keep the notation simple.

The last constructor is LO, short for "linear operator". In $\mathsf{LO}(\mathfrak{h},g)$, the metavariable $\mathfrak{h}$ stands for an *integrand* variable, like h in equation (2). As the typing rule indicates, $\mathfrak{h}$ takes scope over g. We use LO to name a measure by specifying how it integrates a function: $\mathsf{LO}(\mathfrak{h},g)$ means the measure m such that the integral of a measurable nonnegative function $\mathfrak{h}$ with respect to m is g.

An operational way to interpret a measure term is to run it as an *importance sampler*, which generates a random outcome along with a weight. Interpreted

thus, Gaussian means to draw a number from a Gaussian distribution, Ret means to produce the given outcome, and Bind means to sequence two importance samplers. A nonnegative linear combination of measures

$$\mathsf{Msum}\big(\mathsf{Weight}(e_1, m_1), \ldots, \mathsf{Weight}(e_n, m_n)\big) \tag{12}$$

is an n-way random choice: we choose one of the measures m_i with probability proportional to e_i, and at the same time multiply the current weight (which starts at 1) by $\sum_{i=1}^{n} e_i$. In particular, if $m_1 = \cdots = m_n = 1/n$ then we choose m_i uniformly and leave the weight unaffected. That is why equation (4) means to flip a fair coin. If an immediate subterm m_i of $\mathsf{Msum}(m_1, \ldots, m_n)$ is not built with Weight, then we treat m_i like $\mathsf{Weight}(1, m_i)$. And if the measure term $\mathsf{Weight}(e, m)$ occurs in isolation, then its sampling interpretation is as with $\mathsf{Msum}(\mathsf{Weight}(e, m))$: scale the weight by e and continue with m.

We can also interpret a measure term denotationally—as a measure. To define the measure denoted by each measure term, we specify the integral of a function with respect to the measure. Our approach to simplification starts with this definition, so we turn to it next.

3 Connecting abstract and concrete integration

A fundamental result of measure theory is that measures can be viewed in two equivalent ways (Theorems ⟨12⟩ and ⟨13⟩ in Pollard 2001).

On one hand, we can view a measure as a function that maps sets to their sizes in $\mathbb{R}^+$, by *measuring* the sets' length or volume.

On this view, the uniform distribution on the unit interval ($\mathsf{Uniform}(0, 1)$ in our language) is a function that maps the interval $[2/3, 2]$ to the number $1/3$, because $[0, 1] \cap [2/3, 2]$ has length $1/3$. In other words, the probability is $1/3$ for a point randomly drawn from $\mathsf{Uniform}(0, 1)$ to lie in $[2/3, 2]$. The same function maps the singleton set $\{1/2\}$ to the number 0, which is the probability for a point randomly drawn from $\mathsf{Uniform}(0, 1)$ to be exactly $1/2$. Similarly, the *Dirac distribution* at $1/2$ (which we write $\mathsf{Ret}(1/2)$) is a function that maps each set S to 1 if $1/2 \in S$, and to 0 otherwise. So it maps the interval $[2/3, 1]$ to 0 and the singleton $\{1/2\}$ to 1. After all, if we draw the point $1/2$ deterministically, then the probability that the point lies in $[2/3, 1]$ is 0, and the probability that the point is exactly $1/2$ is 1.

On the other hand, we can view a measure as a function that maps functions to their integrals in $\mathbb{R}^+$, with higher-order type $(\cdots \to \mathbb{R}^+) \to \mathbb{R}^+$, so mathematicians call it an *operator*. On this view, $\mathsf{Uniform}(0, 1)$ is a function that maps h to $\int_0^1 h(x)\,dx$. For example, it maps $\lambda x.\, x$ to $\int_0^1 x\,dx = 1/2$. In other words, the expected value of a random draw from $\mathsf{Uniform}(0, 1)$ is $1/2$. We can view the Dirac distribution $\mathsf{Ret}(1/2)$ as a function as well. It maps h to the number $h(1/2)$. And lo and behold, the expected value of drawing the number $1/2$ deterministically is $1/2$.

It may seem strange to call the application $h(1/2)$ an integral of h, but it satisfies the important properties of integration that we use: it preserves nonneg-

ativity and is *linear*:

$$\big(\lambda x.\, a \cdot h(x) + b \cdot k(x)\big)(1/2) = a \cdot h(1/2) + b \cdot k(1/2) \tag{13}$$

for any functions $h, k : \mathbb{R} \to \mathbb{R}^+$ and weights $a, b : \mathbb{R}^+$. Thus the application of a measure m as a linear operator to a function h is called the *(abstract) integral* of the *integrand* h with respect to m. We represent it by $\mathsf{integrate}(m, h)$.

These two views are equivalent, as mentioned previously. More specifically, the size of a set S (with respect to a measure m) is the integral $\mathsf{integrate}(m, \chi_S)$ of its *characteristic function* χ_S, defined by $\chi_S(x) = 1$ if $x \in S$ and $\chi_S(x) = 0$ otherwise.

Despite this correspondence, we need to view a measure as an integrator, for three reasons. First, abstract integration is required to define the Bind operation of the measure monad (Giry 1982), and Bind is essential in the probabilistic programs we seek to simplify. Second, existing CASes can handle concrete integrals, as demonstrated in the introduction. Third, existing CASes are weak at representing and measuring sets.

The ability to compute integrals is useful to us in the common cases where abstract integration involves concrete integration, like with Uniform and Gaussian. But to take advantage of this ability, we need to relate concrete integration to abstract integration, which CASes off the shelf do not even represent. For example, Maple has plenty of facilities we covet for simplifying and transforming concrete integrals $\int$, sums $\sum$, function applications, and their iterated combinations, but the facilities for integration, for summation, and for function application are separate, and there is no single way to represent, say, a product measure.

Hence, our plan is to simplify measure terms in three steps:

1. View the given measure term as a linear operator, which specifies the abstract integral of an arbitrary integrand.
2. Improve the integral using computer algebra, keeping the integrand arbitrary.
3. Read off a new measure term from the improved linear operator.

A major contribution of this paper is to automate not only step 1 (Section 4) but also step 3 (Section 5), so that the task of simplifying probabilistic programs reduces to the task of improving integrals. This reduction leads to another major contribution of this paper, namely to improve integrals using the building blocks provided by an existing CAS (Section 6).

4 From measure term to linear operator

Step 1 is to convert a given measure term m to a term of the form $\mathsf{LO}(\mathfrak{h}, g)$. Recall that the $\mathfrak{h}$ in $\mathsf{LO}(\mathfrak{h}, g)$ represents an arbitrary function to be integrated, so we set $\mathfrak{h}$ to a fresh name. Then, to preserve the meaning of m, the g in $\mathsf{LO}(\mathfrak{h}, g)$ should be the abstract integral of $\mathfrak{h}$ with respect to m, so we set g to $\mathsf{integrate}(m, \mathfrak{h})$, defined in Figure 2.

Figure 2 defines $\mathsf{integrate}(m, h)$ by structural induction on measure term m, so our definition serves as a compositional denotational semantics of measure terms,

$$
\begin{aligned}
&\mathsf{integrate}\big(m, && h\big) = \textstyle\int_{\mathcal{L}(m)}^{\mathcal{U}(m)} \mathcal{D}(m,x)\cdot h(x)\,dx \\
&&& \quad \text{where } m \text{ is a primitive measure term and } x \text{ is fresh} \\
&\mathsf{integrate}\big(\mathsf{Ret}(e), && h\big) = h(e) \\
&\mathsf{integrate}\big(\mathsf{Bind}(m,x,m'), && h\big) = \mathsf{integrate}\big(m, \lambda x.\, \mathsf{integrate}(m', h)\big) \\
&\mathsf{integrate}\big(\mathsf{Msum}(m,\dots), && h\big) = \mathsf{integrate}(m,h) + \cdots \\
&\mathsf{integrate}\big(\mathsf{Weight}(e,m), && h\big) = e \cdot \mathsf{integrate}(m,h) \\
&\mathsf{integrate}\big(\mathsf{If}(e,m,m'), && h\big) = \mathsf{If}\big(e, \mathsf{integrate}(m,h), \mathsf{integrate}(m',h)\big) \\
&\mathsf{integrate}\big(\mathsf{LO}(\mathfrak{h},g), && h\big) = g\{\mathfrak{h} \mapsto h\} \\
&\mathsf{integrate}\big(m, && h\big) = \mathsf{Integrate}(m,h) && \text{otherwise}
\end{aligned}
$$

Fig. 2. Translating a measure term m to a linear operator $\mathsf{LO}(\mathfrak{h}, \mathsf{integrate}(m,\mathfrak{h}))$, where $\mathfrak{h}$ is fresh

Table 1. Defining properties of primitive measures

Measure term	Bounds		Density	Holonomic representation
m	$\mathcal{L}(m)$	$\mathcal{U}(m)$	$\mathcal{D}(m,x)$	$\frac{p_0(x)}{p_1(x)} = -\frac{\frac{d}{dx}\mathcal{D}(m,x)}{\mathcal{D}(m,x)}$
$\mathsf{Uniform}(a,b)$	a	b	$\frac{1}{b-a}$	0
$\mathsf{Gaussian}(\mu,\sigma)$	$-\infty$	$+\infty$	$\frac{\exp\left(-\frac{1}{2}\cdot\left(\frac{x-\mu}{\sigma}\right)^2\right)}{\sqrt{2\cdot\pi}\cdot\sigma}$	$\frac{x-\mu}{\sigma^2}$
$\mathsf{Cauchy}(\mu,\gamma)$	$-\infty$	$+\infty$	$\frac{\left(1+\left(\frac{x-\mu}{\gamma}\right)^2\right)^{-1}}{\pi\cdot\gamma}$	$\frac{2\cdot(x-\mu)}{(x-\mu)^2+\gamma^2}$
$\mathsf{StudentT}(\nu,\mu,\gamma)$	$-\infty$	$+\infty$	$\frac{\left(1+\left(\frac{x-\mu}{\gamma}\right)^2\nu\right)^{-(\nu+1)/2}}{\frac{\Gamma(\nu/2)}{\Gamma((\nu+1)/2)}\cdot\sqrt{\pi\cdot\nu}\cdot\gamma}$	$\frac{(\nu+1)\cdot(x-\mu)}{(x-\mu)^2+\gamma^2\cdot\nu}$
$\mathsf{Beta}(\alpha,\beta)$	0	1	$\frac{x^{\alpha-1}\cdot(1-x)^{\beta-1}}{\mathrm{B}(\alpha,\beta)}$	$\frac{(\alpha+\beta-2)\cdot x-(\alpha-1)}{x\cdot(1-x)}$
$\mathsf{Gamma}(k,\theta)$	0	$+\infty$	$\frac{x^{k-1}\cdot\exp(-\frac{x}{\theta})}{\Gamma(k)\cdot\theta^k}$	$\frac{\frac{x}{\theta}+1-k}{x}$

and a standard one at that. The integrand h is an accumulator argument that may not just be a name $\mathfrak{h}$, as can be seen in the right-hand side of the Bind case. Actually, h amounts to a continuation, so step 1 amounts to a one-pass transform from monadic to continuation-passing style (CPS) (Hatcliff and Danvy 1994) (more precisely, to continuation-composing style (Danvy and Filinski 1990)).

When m is primitive, integrate produces a concrete integral using the standard properties of m listed in Table 1. In this paper, all primitive measures range over real intervals. The properties we use from Table 1 are

– $\mathcal{L}(m)$, the lower bound of the interval, possibly $-\infty$;

– $\mathcal{U}(m)$, the upper bound of the interval, possibly $+\infty$; and
– $\mathcal{D}(m, x)$, the *density* of m with respect to the Lebesgue measure.

Many more measures could be added to Table 1, but some measures cannot be so represented, such as $\mathsf{Ret}(0)$ and the logistic distribution.

The case where the input measure term m is a free variable representing an unknown measure is handled by the last line in Figure 2, which produces a residual term $\mathsf{Integrate}(m, h)$.

For the example (1), step 1 produces the term

$$\mathsf{LO}\left(h, \int_0^1 \frac{1}{1-0} \cdot \int_0^1 \frac{1}{1-0} \cdot \left(\begin{cases} h(\mathsf{true}) & \text{if } x < y \\ h(\mathsf{false}) & \text{otherwise} \end{cases}\right) dy\, dx\right), \tag{14}$$

in which we notate the conditional If with a curly left brace. Maple immediately removes the factor $\frac{1}{1-0}$, yielding equation (2).

As equation (14) illustrates, the output of this integrate step is patently a linear operator. In other words, just by examining the integral produced syntactically, without any deep reasoning, we can tell that it is linear in the integrand $\mathfrak{h}$. Formally, we say that g is *patently linear in* $\mathfrak{h}$ iff g is generated by the grammar

$$g ::= \mathfrak{h}(e) \mid g_1 + \cdots + g_n \mid \int_a^b g\, dx \mid e \cdot g \mid \mathsf{If}(e, g, g') \mid \mathsf{Integrate}(m, \lambda x.\, g) \tag{15}$$

where a, b, e, m do not contain $\mathfrak{h}$ free, and $x \neq \mathfrak{h}$. (For simplicity, this grammar omits $\sum_a^b g\, dx$, and this paper omits measures over $\mathbb{Z}$.)

Of course, patent linearity entails linearity, but the converse is not the case. For example, the term $\sin(h(a))^2 + \cos(h(a))^2 - 1$ is linear in h (because it is zero) but not patently linear. Still, the right-hand sides in Figure 2 all maintain patent linearity, so step 1 produces a patently linear integral, which we then subject to algebraic manipulations in step 2. As long as our manipulations of the integral preserve patent linearity, the result can be turned back to a measure term. That is the job of step 3, which we present next.

5 From linear operator back to measure term

Figure 3 defines $\mathsf{unintegrate}(\mathfrak{h}, g, c)$, whose goal is to find a measure term m (built without LO) such that $\mathsf{integrate}(m, \mathfrak{h}) = g$. If we think of step 1 above as a transform from monadic style to CPS, then step 3 inverts this.

Like integrate, unintegrate proceeds by structural induction on the input term. Most lines of this definition just handle a case in the grammar (15) by inverting a corresponding line defining integrate in Figure 2. The main deviations from this pattern are three, described in the three subsections below.

5.1 Peephole optimizations

As unintegrate builds a measure term, it invokes the smart constructors bind and weight, defined in Figure 3. They are semantically equivalent to Bind and Weight but perform peephole optimizations using algebraic laws: bind uses the right and

$$
\begin{aligned}
\mathsf{unintegrate}\big(\mathfrak{h}, \textstyle\int_a^b g\,dx, \quad c\big) &= \mathsf{weight}\big(e_1, \mathsf{bind}(m, x, \mathsf{weight}(e_2, m'))\big) \\
&\quad \text{where } x \neq \mathfrak{h} \text{ and } (e, m') = \mathsf{unweight}\big(\mathsf{unintegrate}(\mathfrak{h}, g, c \wedge a < x < b)\big) \\
&\quad\quad (m, e') = \mathsf{recognize}(e, x, a, b, c) \\
&\quad\quad e_1 \cdot e_2 = e' \qquad \text{where } e_1 \text{ does not contain } x \text{ free} \\
\mathsf{unintegrate}\big(\mathfrak{h}, \mathfrak{h}(e), \quad c\big) &= \mathsf{Ret}(e) \\
\mathsf{unintegrate}\big(\mathfrak{h}, g + \cdots, \quad c\big) &= \mathsf{Msum}\big(\mathsf{unintegrate}(\mathfrak{h}, g, c), \ldots\big) \\
\mathsf{unintegrate}\big(\mathfrak{h}, e \cdot g, \quad c\big) &= \mathsf{weight}\big(e, \mathsf{unintegrate}(\mathfrak{h}, g, c)\big) \\
&\quad\quad \text{where } e \text{ does not contain } \mathfrak{h} \text{ free} \\
\mathsf{unintegrate}\big(\mathfrak{h}, \mathsf{If}(e, g, g'), \quad c\big) &= \mathsf{If}\big(e, \mathsf{unintegrate}(\mathfrak{h}, g, c \wedge e), \mathsf{unintegrate}(\mathfrak{h}, g', c \wedge \neg e)\big) \\
\mathsf{unintegrate}\big(\mathfrak{h}, \mathsf{Integrate}(m, h), c\big) &= \mathsf{bind}\big(m, x, \mathsf{unintegrate}(\mathfrak{h}, h(x), c)\big) \quad \text{where } x \text{ is fresh} \\
\mathsf{unintegrate}\big(\mathfrak{h}, g, \quad c\big) &= \mathsf{LO}(\mathfrak{h}, g) \qquad \text{otherwise}
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{bind}(m, \quad x, \mathsf{Ret}(x)) &= m & \mathsf{weight}(1, m) &= m \\
\mathsf{bind}(\mathsf{Ret}(e), x, m) &= m\{x \mapsto e\} & \mathsf{weight}(0, m) &= \mathsf{Msum}() \\
\mathsf{bind}(m, \quad x, m') &= \mathsf{Bind}(m, x, m') & \mathsf{weight}(e, \mathsf{Weight}(e', m)) &= \mathsf{weight}(e \cdot e', m) \\
&\quad \text{otherwise} & \mathsf{weight}(e, m) &= \mathsf{Weight}(e, m) \\
& & &\quad \text{otherwise}
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{unweight}\big(\mathsf{Weight}(e, m)\big) &= \big(e, m\big) \\
\mathsf{unweight}\big(\mathsf{Msum}(m_1, \ldots, m_n)\big) &= \big(e, \mathsf{Msum}(\mathsf{weight}(\tfrac{1}{e}, m_1), \ldots, \mathsf{weight}(\tfrac{1}{e}, m_n))\big) \\
&\quad \text{where } e = e_1 + \cdots + e_n \text{ and } (e_i, \_) = \mathsf{unweight}(m_i) \\
\mathsf{unweight}\big(m\big) &= \big(1, m\big) \qquad \text{otherwise}
\end{aligned}
$$

Fig. 3. Translating a linear operator $\mathsf{LO}(\mathfrak{h}, g)$ to a measure term $\mathsf{unintegrate}(\mathfrak{h}, g, \mathsf{true})$

left monad identity laws to eliminate Bind, and weight uses linearity to eliminate Weight. We wait until step 3 to perform these optimizations, so that they apply to terms produced by density recognition (Section 5.3).

5.2 Assumption context

As unintegrate traverses the input integral g, it maintains a *context* of assumptions about the variables in g. For example, when unintegrate processes $\mathsf{If}(e, g, g')$, it conjoins the condition e onto the current context c while processing the then-case g, and it conjoins $\neg e$ onto c while processing the else-case g'. So if e is the condition $x \geq 0$, then the recursive call to $\mathsf{unintegrate}(\mathfrak{h}, g, c \wedge e)$ is free to reduce any occurrence of $\sqrt{x^2}$ to x. Similarly, when unintegrate processes $\int_a^b g\,dx$, it conjoins $a < x < b$ onto the current context c. (Actually that recursive call to $\mathsf{unintegrate}(\mathfrak{h}, g, c \wedge a < x < b)$ is free to ignore any countable number of points in the interval $[a, b]$, but unintegrate does not currently exercise that freedom.)

5.3 Density recognition

When unintegrate processes $\int_a^b g\,dx$, it tries to recognize a concrete integral that belongs to a primitive measure. For example, when unintegrate processes the con-

crete integral in equation (2), it recognizes $\int_0^1 \ldots dx$ as belonging to $\mathsf{Uniform}(0,1)$ and so produces a term of the form $\mathsf{Bind}(\mathsf{Uniform}(0,1), x, \ldots)$. This requires nontrivial computer algebra because there is a gulf between the body of that integral, namely the integral $\int_0^1 \left(\begin{cases} h(\mathsf{true}) & \text{if } x < y \\ h(\mathsf{false}) & \text{otherwise} \end{cases}\right) dy$, and $\mathcal{D}(m,x) \cdot h(x)$ at the top of Figure 2, namely the product $\frac{1}{b-a} \cdot h(x)$. These terms do not unify syntactically.

For recognizing primitive measures other than $\mathsf{Uniform}$, the gulf is even wider, and syntactic matching is even more futile. For example, how can an *algorithm* recognize that the density in (9) is proportional to a certain Gaussian distribution and read off the measure term (10)? When $\mathsf{unintegrate}$ processes the integral (9), the recursive call to $\mathsf{unintegrate}(\mathfrak{h}, g, c')$ returns the measure term

$$\mathsf{Weight}\left(\frac{\exp\left(-\frac{x^2}{2}\right)}{\sqrt{2\cdot\pi}} \cdot \frac{\exp\left(-\frac{(y-x)^2}{2}\right)}{\sqrt{2\cdot\pi}},\ \mathsf{Ret}(x)\right), \tag{16}$$

and the call to $\mathsf{unweight}$ (defined in Figure 3) decomposes this term into its two subterms, the weight

$$f(x) = \frac{\exp\left(-\frac{x^2}{2}\right)}{\sqrt{2\cdot\pi}} \cdot \frac{\exp\left(-\frac{(y-x)^2}{2}\right)}{\sqrt{2\cdot\pi}} \tag{17}$$

and the measure $\mathsf{Ret}(x)$. The weight is a function in the integration variable x. We call this function the *target density*. We need an algorithm to recognize that it is proportional to $\mathcal{D}\big(\mathsf{Gaussian}(\mu,\sigma)\big)$ and not, say, $\mathcal{D}\big(\mathsf{Cauchy}(\mu,\gamma)\big)$.

Holonomic representation We solve this density recognition problem in one fell swoop for *all* our primitive measures, by treating the target density f as a *holonomic*[3] expression (Wilf and Zeilberger 1992; Chyzak and Salvy 1998). That means we find a *homogeneous linear differential equation*

$$p_n(x) \cdot f^{(n)}(x) + \cdots + p_1(x) \cdot f'(x) + p_0(x) \cdot f(x) = 0 \tag{18}$$

that defines $f(x)$ up to a constant factor, in which each $p_i(x)$ is a polynomial in x. For example, the density (17) is defined up to a constant factor by

$$1 \cdot f'(x) + (2 \cdot x - y) \cdot f(x) = 0, \tag{19}$$

so $n = 1$, $p_1(x) = 1$, and $p_0(x) = 2 \cdot x - y$.

The general algorithm for finding a differential equation (18) from a closed-form expression $f(x)$ is well established and efficiently implemented by the Maple function `gfun[holexprtodiffeq]`[4] (Salvy and Zimmermann 1994). Such differential equations are only determined up to scaling by a polynomial in x. For example, scaling (19) by x yields another differential equation

$$x \cdot f'(x) + (2 \cdot x^2 - y \cdot x) \cdot f(x) = 0, \tag{20}$$

[3] More precisely, a D-finite expression, but we will use holonomicity more generally when dealing with discrete densities also.

[4] The version of `gfun` shipped with Maple has several important bugs, fixed in the version at `http://perso.ens-lyon.fr/bruno.salvy/software/the-gfun-package/`

which is also satisfied by (17), as well as by some new but singular solutions (which can be found by Laplace-transform methods). To reduce this degree of freedom and eliminate such spurious solutions, we divide the differential equation by the leading coefficient $p_n(x)$, so that the coefficients become the rational functions $1, p_{n-1}(x)/p_n(x), \ldots, p_0(x)/p_n(x)$. We then use the non-leading coefficients to identify the primitive measure.

For all our primitive measures (listed in Table 1), it turns out that $n = 1$ (in other words, the differential equation is first-order), so we have just one ratio $p_0(x)/p_1(x)$ to consider. For the density (17), this ratio is $2 \cdot x - y$. In general, when $n = 1$, this ratio is equal to $-f'(x)/f(x)$, but the algorithm that computes this ratio is completely different from differentiating f then dividing by f and hoping that the factors that do not form a rational function cancel out. Rather, the algorithm achieves efficiency and robustness by performing linear algebra on *Ore algebra* elements, which represent linear differential operators and enjoy many closure properties that are efficiently computable (Salvy 2005).

Rational-function matching Thanks to existing efficient algorithms for normalizing rational functions (such as Euclid's algorithm), we can easily and robustly match rational functions against each other, and thus recognize all our primitive measures in the same way. For example, given that $n = 1$ and the ratio $p_0(x)/p_1(x) = 2 \cdot x - y$ is linear in x for our target density (17), we look through the rightmost column of Table 1 for a rational function whose numerator's degree in x is one higher than its denominator's.[5] The only candidate is $m = \mathsf{Gaussian}(\mu, \sigma)$, and its integration bounds match those of (9), so we equate corresponding coefficients ($2 = 1/\sigma^2$ and $-y = -\mu/\sigma^2$) and solve to get $\mu = y/2$ and $\sigma = 1/\sqrt{2}$. This solving step can be done by calling Maple's `solve` or by designing a custom matcher for each primitive measure. We prefer the latter as it is more efficient.

The recognize algorithm In general, unintegrate in Figure 3 tries to recognize a target density e by invoking $\mathsf{recognize}(e, x, a, b, c)$. Here the density expression e may contain the integration variable x free, a and b are the bounds on x, and c is the context. The goal of recognize is to robustly satisfy the equation

$$\mathsf{recognize}\big(\mathcal{D}(m, x) \cdot e', x, \mathcal{L}(m), \mathcal{U}(m), c\big) = (m, e'), \tag{21}$$

where m is a primitive measure term and e' does not contain x free. To do so, recognize proceeds through three steps:

1. Convert e as a function of x to a holonomic representation.
2. Find the primitive measure m by rational-function matching as described above, under the assumption context c. Thus, new primitive measures can be added to be recognized simply by extending Table 1.
3. Solve for the constant factor e', by equating the target e and the matched $\mathcal{D}(m, x) \cdot e'$ at certain points x. To choose x, either use initial conditions returned by `gfun[holexprtodiffeq]`, if any, or default to $x \in \{0, 1/2, 1\}$.

[5] We only consider the difference between the two degrees, because the numerator and denominator may not be relatively prime, like for $\mathsf{Beta}(1, \beta)$ and $\mathsf{Beta}(\alpha, 1)$.

If any step above fails, then recognize resorts to the following:

- if $-\infty < a < b < \infty$, then return $\big(\mathsf{Uniform}(a,b), e_1 \cdot (b-a)\big)$;
- otherwise, return $\big(\mathsf{LO}(\mathfrak{h}', \int_a^b \mathfrak{h}'(x)\,dx), e_1\big)$, where $\mathfrak{h}'$ is fresh;

even though e_1 may contain x free. Returning LO is our unobtrusive way to admit failure, which is also done in the the catch-all case at the bottom of Figure 3 when g is not patently linear.

6 Improving linear operators algebraically

The previous two sections established a two-way bridge between measure terms and patently linear expressions. Patently linear expressions are real-valued expressions, and computer algebra gives us many more tools for these than for measure terms. We now put these tools to work.

To start with, every CAS incessantly performs so-called *automatic simplification* on every expression, using linear-time algorithms that are much faster than naive rewrite rules. Automatic simplification ensures that

- addition $+$ and multiplication $\cdot$ are commutative and associative, with identities 0 and 1, so for example $e_1 \cdot (1 \cdot e_2 \cdot g)$ is equivalent to $(e_1 \cdot e_2) \cdot g$;
- repeated terms are collected, so for example $g_1 + (g_1 + g_2)$ becomes $2 \cdot g_1 + g_2$;
- arithmetic on rational numbers happens, so the factor $\frac{1}{1-0}$ in (14) disappears.

Our language of patently linear expressions take advantage of automatic simplification pervasively. In particular, in Figures 3 and 4, we pattern-match against $+$ and $\cdot$ using commutativity and associativity. Hence just by composing steps 1 and 3 (Figures 2 and 3), we already simplify the measure term

$$\mathsf{Weight}(e_1, \mathsf{Weight}(1, \mathsf{Weight}(e_2, \mathsf{Msum}(m_1, \mathsf{Msum}(m_1, m_2))))) \tag{22}$$

to $\mathsf{Weight}(e_1 \cdot e_2, \mathsf{Msum}(\mathsf{Weight}(2, m_1), m_2))$.

This immediate success of automatic simplification might embolden us to feed patently linear expressions to Maple functions such as `value`, which triggers symbolic integration, and `simplify`, which who knows what it does (Moses 1971; Carette 2004). But such optimism would be misplaced. First, CASes tend to be ineffective at multidimensional integrals, which are typical of probabilistic programs. For example, Maple's `value` can only simplify the double integral (6) (to (7)) if we reverse the order of integration between x and y. On the other hand, even though there is no hope to improve a nested integral of the form

$$\iiint \exp(x \cdot y \cdot z) \cdot h(x,y,z)\,dz\,dy\,dx \tag{23}$$

because h is arbitrary, the amount of time Maple's `value` takes to return the integral unimproved explodes as the nesting depth increases.

Second, the nebulous notion of simplification that CASes today come with can wreck havoc on patently linear expressions. For example, consider this probabilistic program, which draws a number from a Gaussian distribution then returns its absolute value:

$$\mathsf{Bind}(\mathsf{Gaussian}(0,1), x, \mathsf{If}(x < 0, \mathsf{Ret}(-x), \mathsf{Ret}(x))) \tag{24}$$

In step 1, we compute the corresponding linear operator

$$\mathsf{LO}\left(h, \int_{-\infty}^{\infty} \frac{\exp\left(-\frac{x^2}{2}\right)}{\sqrt{2\cdot\pi}} \cdot \left(\begin{cases} h(-x) & \text{if } x < 0 \\ h(x) & \text{otherwise} \end{cases}\right) dx\right). \tag{25}$$

If we feed this linear operator straight to step 3, then we recover the measure term (24) unchanged. But if we feed it to Maple's `simplify` or `value`, the Gaussian density becomes duplicated and unrecognized: we get the linear operators

$$\mathsf{LO}\left(h, \int_{-\infty}^{\infty} \left(\begin{cases} \frac{\exp(-\frac{x^2}{2})}{\sqrt{2\cdot\pi}} \cdot h(-x) & \text{if } x < 0 \\ \frac{\exp(-\frac{x^2}{2})}{\sqrt{2\cdot\pi}} \cdot h(x) & \text{otherwise} \end{cases}\right) dx\right) \tag{26}$$

$$\mathsf{LO}\left(h, \int_{-\infty}^{0} \frac{\exp\left(-\frac{x^2}{2}\right)}{\sqrt{2\cdot\pi}} \cdot h(-x)\, dx + \int_{0}^{\infty} \frac{\exp\left(-\frac{x^2}{2}\right)}{\sqrt{2\cdot\pi}} \cdot h(x)\, dx\right), \tag{27}$$

which step 3 cannot express as measure terms without LO. Therefore, trying to improve a patently linear expression by feeding it willy-nilly to symbolic integration or simplification can be not only ineffective but even counter-productive.

We achieve effective improvement by *controlled* application of symbolic integration and algebraic simplification. Informally, our idea is to apply these CAS facilities throughout a patently linear expression *except* where the grammar (15) calls for a patently linear subexpression. Intuitively, these places form the control-flow tree of a probabilistic program. For example, in (25) we withhold the conditional If subexpression from CAS facilities, and in (6) we withhold $h(y)$.

Our workhorse is the reduce function defined in Figure 4. Starting with a linear operator $\mathsf{LO}(\mathfrak{h}, g)$, the goal of $\mathsf{reduce}(\mathfrak{h}, g, c)$ is to improve g under the assumption context c (initially true, as in Section 5.2). The reduce function exercises control over CAS facilities by following the grammar (15). To start with, reduce only invokes $\mathsf{simplify}(e, c)$ in three places, namely where the grammar (15) calls for a general e rather than some patently linear g. The call $\mathsf{simplify}(e, c)$ means to simplify the expression e under the context c (using Maple's `simplify`).

When reduce encounters an integral $\int_a^b g\, dx$, it first checks if the integration variable x is used in any call to the arbitrary integrand $\mathfrak{h}$ (in other words, if x is used to Return any outcome). If not, then the variable x could be *eliminated* (Dechter 1998) or *integrated out*. An example is the variable x in (6). To perform the integral over x, we push it in past the integral over y, to form the expression

$$\int_{-\infty}^{\infty} \left(\int_{-\infty}^{\infty} \frac{\exp\left(-\frac{x^2}{2}\right)}{\sqrt{2\cdot\pi}} \cdot \frac{\exp\left(-\frac{(y-x)^2}{2}\right)}{\sqrt{2\cdot\pi}}\, dx \right) \cdot h(y)\, dy, \tag{28}$$

$$\mathsf{reduce}\big(\mathfrak{h}, \textstyle\int_a^b g\,dx,\quad c\big) = \begin{cases} \mathsf{reduce}(\mathfrak{h}, g', c) \\ \quad \text{if no argument to } \mathfrak{h} \text{ in } g \text{ contains } x \text{ free} \\ \quad\quad \mathfrak{h}' \text{ is fresh} \\ \quad\quad g' = \mathsf{banish}\big(\mathsf{LO}(\mathfrak{h}', \textstyle\int_a^b \mathfrak{h}'(x)\,dx), x, \mathfrak{h}, g\big) \\ \quad\quad g' \text{ contains fewer } \textstyle\int \text{ signs than } \int_a^b g\,dx \text{ does} \\ 0 \quad \text{if } c' \text{ is inconsistent} \\ \mathsf{if}\big(c_1', \cdot \textstyle\int_{a'}^{b'} \mathsf{if}(c_2', g', 0)\,dx, 0\big) \\ \quad \text{if } c_1' \wedge c_2' \wedge a' < x < b' \text{ is equivalent to } c' \\ \quad\quad c_1' \text{ does not contain } x \text{ free} \end{cases}$$

$$\text{where } x \neq \mathfrak{h} \text{ and } \mathsf{Indicator}(c_1) \cdot \cdots \cdot \mathsf{Indicator}(c_n) \cdot g' = \mathsf{reduce}(\mathfrak{h}, g, c \wedge a < x < b)$$
$$c' = c_1 \wedge \cdots \wedge c_n \wedge a < x < b$$

$$\mathsf{reduce}\big(\mathfrak{h}, g + \cdots,\quad c\big) = \mathsf{reduce}(\mathfrak{h}, g, c) + \cdots$$

$$\mathsf{reduce}\big(\mathfrak{h}, e \cdot g,\quad c\big) = \mathsf{simplifyIf}\big(\mathsf{simplify}(e, c)\big) \cdot \mathsf{reduce}(\mathfrak{h}, g, c) \quad \text{where } e \text{ does not contain } \mathfrak{h} \text{ free}$$

$$\mathsf{reduce}\big(\mathfrak{h}, \mathsf{If}(e, g, g'),\quad c\big) = \mathsf{if}\big(\mathsf{simplify}(e, c), \mathsf{reduce}(\mathfrak{h}, g, c \wedge e), \mathsf{reduce}(\mathfrak{h}, g', c \wedge \neg e)\big)$$

$$\mathsf{reduce}\big(\mathfrak{h}, \mathsf{Integrate}(m, h), c\big) = \mathsf{Integrate}\big(m, \lambda x.\, \mathsf{reduce}(\mathfrak{h}, h(x), c)\big) \quad \text{where } x \text{ is fresh}$$

$$\mathsf{reduce}\big(\mathfrak{h}, g,\quad c\big) = \mathsf{simplify}(g, c) \quad \text{otherwise}$$

$$\mathsf{simplifyIf}(\mathsf{If}(c, e, e')) = \mathsf{if}(c, \mathsf{simplifyIf}(e), \mathsf{simplifyIf}(e'))$$
$$\mathsf{simplifyIf}(e) = e \quad \text{otherwise}$$

Fig. 4. Improving a linear operator $\mathsf{LO}(\mathfrak{h}, g)$ to $\mathsf{LO}(\mathfrak{h}, \mathsf{reduce}(\mathfrak{h}, g, \mathsf{true}))$. The auxiliary functions banish and if are defined in Figure 5.

then ask Maple to do just the inner integral over x, which no longer contains any call to h. In general, we push the integral to be eliminated all the way in, so that it becomes the innermost integral and does not contain any call to $\mathfrak{h}$, then ask Maple to do just that integral. This pushing and symbolic integration is performed by banish, defined in Figure 5 in the appendix. The specification of banish is that $\mathsf{LO}(\mathfrak{h}, \mathsf{banish}(m, x, \mathfrak{h}, g))$ should mean the same measure as $\mathsf{Bind}(m, x, \mathsf{LO}(\mathfrak{h}, g))$ but integrate over x innermost rather than outermost. Achieving this specification requires doubling the work when g is an If. The upshot of this improvement is that we manage to simplify the measure term (5) to $\mathsf{Gaussian}(0, \sqrt{2})$.

If the integral $\int_a^b g\,dx$ cannot be eliminated altogether, then reduce tries to shrink the integral's bounds a, b by checking g to see if it looks like $\mathsf{If}(a' < x < b', g', 0)$ or $\mathsf{If}(a' < x < b', 1, 0) \cdot g'$. If it does, then the recursive call to $\mathsf{reduce}(\mathfrak{h}, g, c \wedge a < x < b)$ would return a product like $\mathsf{Indicator}(a' < x < b') \cdot g'$. The meaning of $\mathsf{Indicator}(\ldots)$ is same as $\mathsf{If}(\ldots, 1, 0)$, but we introduce a separate constructor Indicator for this particular use of If to encode a domain restriction that reduce should incorporate into the integration bounds. The constructor Indicator is called in the appropriate cases by the smart constructor if, defined in Figure 5 in the appendix. Any domain restriction so gathered is then used by reduce to shrink

the integration bounds. The upshot of this improvement is that we manage to simplify the measure terms

$$\mathsf{Bind}(\mathsf{Uniform}(0,1), x, \mathsf{If}(0 < x < 1/2, \mathsf{Ret}(x), \mathsf{Msum}())) \tag{29}$$

$$\mathsf{Bind}(\mathsf{Uniform}(0,1), x, \mathsf{Weight}(\mathsf{If}(0 < x < 1/2, 1, 0), \mathsf{Ret}(x))) \tag{30}$$

both to $\mathsf{Weight}(1/2, \mathsf{Uniform}(0, 1/2))$.

Besides these improvements that constitute reduce, we have found two other operations on patently linear expressions to be sometimes useful: pushing an integral inward but not all the way in; and reparameterizing an integral so that the integration variable is what's passed to the arbitrary integrand $\mathfrak{h}$. These operations are not always beneficial, so reduce does not perform them automatically; rather, human or heuristic guidance is required to invoke them for now.

7 Evaluation

Through our work on probabilistic programming, we have assembled a test suite with 66 input-output pairs of measure terms, including probabilistic models and inference procedures that arise in practice. Our simplifier passes all of these tests. Many tests have the same input and output, because our simplifier should not degrade any term, especially one that is already as simple as can be. As explained in Section 6, this requirement—although basic—is not trivial to satisfy. Indeed, if we only apply step 1 followed by step 3, 42 of our tests still pass.

Our test suite can be seen online[6] alongside our simplifier implementation,[7] but the following ablation results give a sense of the coverage and variety of our tests. If we remove banish, 14 tests fail. If we remove assumption contexts, 8 tests fail. If we remove calls to Maple's `simplify`, 8 tests fail. If we remove shrinking integration bounds, 9 tests fail. Lastly, if we remove density recognition from step 3, fully 53 tests fail.

Compared to our earlier (unpublished) attempts at simplifying probabilistic programs using computer algebra, this version is quite compact: 775 lines of Maple code (including comments and testing infrastructure). More importantly, the design of this version can be explained, as this paper demonstrates. Especially effective and robust are our conversion between measure terms and linear operators and our use of differential equations for density recognition. These techniques pave the way for two pressing future tasks:

- Add arrays. This will let us express many more distributions in wide use. However, it is a wide-open problem how to deal with product measures and their linear-operator counterparts, especially when their length is symbolic.
- Recover domain restrictions from more advanced uses of If, such as multivariate conditions. Potential solutions include SMT solvers and methods based on normal forms for inequalities.

[6] `https://github.com/hakaru-dev/hakaru/blob/master/maple/NewSLOTests.mpl`
[7] `https://github.com/hakaru-dev/hakaru/blob/master/maple/NewSLO.mpl`

References

Jacques Carette. Understanding expression simplification. In Jaime Gutierrez, editor, *Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 72–79, New York, 2004. ACM Press.

Jacques Carette. A canonical form for piecewise defined functions. In Dongming Wang, editor, *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 77–84, New York, 2007. ACM Press.

Frédéric Chyzak and Bruno Salvy. Non-commutative elimination in Ore algebras proves multivariate holonomic identities. *Journal of Symbolic Computation*, 26(2):187–227, 1998.

Olivier Danvy and Andrzej Filinski. Abstracting control. In *LFP'90: Proceedings of the 1990 ACM Conference on LISP and Functional Programming*, pages 151–160, New York, 1990. ACM Press.

Rina Dechter. Bucket elimination: A unifying framework for probabilistic inference. In Michael I. Jordan, editor, *Learning and Inference in Graphical Models*. Kluwer, Dordrecht, 1998. Paperback: *Learning in Graphical Models*, MIT Press.

Michèle Giry. A categorical approach to probability theory. In Bernhard Banaschewski, editor, *Categorical Aspects of Topology and Analysis: Proceedings of an International Conference Held at Carleton University, Ottawa, August 11–15, 1981*, number 915 in Lecture Notes in Mathematics, pages 68–85, Berlin, 1982. Springer.

John Hatcliff and Olivier Danvy. A generic account of continuation-passing styles. In *POPL'94: Proceedings of the 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 458–471, New York, 1994. ACM Press.

Peter S. Maybeck. *Stochastic Models, Estimation, and Control*. Number 141 in Mathematics in Science and Engineering. Academic Press, San Diego, CA, 1979.

Joel Moses. Algebraic simplification: A guide for the perplexed. *Communications of the ACM*, 14(8):548–560, 1971.

David Pollard. *A User's Guide to Measure Theoretic Probability*. Cambridge University Press, Cambridge, 2001.

Bruno Salvy. D-finiteness: Algorithms and applications. In Manuel Kauers, editor, *Proceedings of the 2005 International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 2–3, New York, 2005. ACM Press.

Bruno Salvy and Paul Zimmermann. Gfun: a Maple package for the manipulation of generating and holonomic functions in one variable. *ACM Transactions on Mathematical Software*, 20(2):163–177, 1994.

Herbert S. Wilf and Doron Zeilberger. An algorithmic proof theory for hypergeometric (ordinary and "q") multisum/integral identities. *Inventiones mathematicae*, 108:557–633, 1992.

$$
\begin{aligned}
&\mathsf{banish}\,\big(m, x, \mathfrak{h}, 0\big) && = 0\\
&\mathsf{banish}\,\big(m, x, \mathfrak{h}, g\big) && = \big(\text{do the integral in } \mathsf{integrate}(m, \lambda x.\, 1)\big) \cdot g\\
& && \quad \text{if } g \text{ does not contain } x \text{ free}\\
&\mathsf{banish}\Big(m, x, \mathfrak{h}, \int_a^b g\, dy\Big) && = \begin{cases} \mathsf{banish}\big(m, x, \mathfrak{h}, \int_{-\infty}^b \mathsf{If}(a < y, g, 0)\, dy\big) \\ \qquad\qquad \text{if } a \text{ contains } x \text{ free} \\ \mathsf{banish}\big(m, x, \mathfrak{h}, \int_a^\infty \mathsf{If}(y < b, g, 0)\, dy\big) \\ \qquad\qquad \text{if } b \text{ contains } x \text{ free} \\ \int_a^b \mathsf{banish}(m, x, \mathfrak{h}, g)\, dy \quad \text{otherwise} \end{cases}\\
& && \quad \text{where } a, b \text{ do not contain } \mathfrak{h} \text{ free and } x \neq \mathfrak{h}\\
& && \qquad\quad m \text{ does not contain } y \text{ free and } x \neq y\\
&\mathsf{banish}\,\big(m, x, \mathfrak{h}, g + \cdots\big) && = \mathsf{banish}(m, x, \mathfrak{h}, g) + \cdots\\
&\mathsf{banish}\,\big(m, x, \mathfrak{h}, e \cdot g\big) && = \mathsf{banish}\big(\mathsf{Bind}(m, x, \mathsf{Weight}(e, \mathsf{Ret}(x))), x, \mathfrak{h}, g\big)\\
& && \quad \text{where } e \text{ does not contain } \mathfrak{h} \text{ free}\\
&\mathsf{banish}\,\big(m, x, \mathfrak{h}, \mathsf{If}(e, g, g')\big) && = \begin{cases} \mathsf{banish}\big(\mathsf{Bind}(m, x, \mathsf{If}(e, \mathsf{Ret}(x), \mathsf{Msum}())), x, \mathfrak{h}, g\big) + \\ \mathsf{banish}\big(\mathsf{Bind}(m, x, \mathsf{If}(e, \mathsf{Msum}(), \mathsf{Ret}(x))), x, \mathfrak{h}, g'\big) \\ \qquad\qquad \text{if } e \text{ contains } x \text{ free} \\ \mathsf{If}\big(e, \mathsf{banish}(m, x, \mathfrak{h}, g), \\ \qquad \mathsf{banish}(m, x, \mathfrak{h}, g')\big) \quad \text{otherwise} \end{cases}\\
&\mathsf{banish}\,\big(m, x, \mathfrak{h}, \mathsf{Integrate}(n, h)\big) && = \mathsf{Integrate}\big(n, \lambda y.\, \mathsf{banish}(m, x, \mathfrak{h}, h(y))\big)\\
& && \quad \text{if } n \text{ does not contain } x \text{ free}\\
& && \quad \text{where } m \text{ does not contain } \mathfrak{h} \text{ free and } y \text{ is fresh}\\
&\mathsf{banish}\,\big(m, x, \mathfrak{h}, g\big) && = \text{do the integral in } \mathsf{integrate}(m, \lambda x.\, g) \text{ otherwise}
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{if}\big(\mathsf{true},\ e, e'\big) && = e\\
&\mathsf{if}\big(\mathsf{false},\ e, e'\big) && = e'\\
&\mathsf{if}\big(c,\ \ e, e\big) && = e\\
&\mathsf{if}\big(c,\ \ e, 0\big) && = \mathsf{Indicator}(c) \cdot e\\
&\mathsf{if}\big(c,\ \ 0, e\big) && = \mathsf{Indicator}(\neg c) \cdot e\\
&\mathsf{if}\big(c,\ \ e, \mathsf{Indicator}(c_1) \cdot \cdots \cdot \mathsf{Indicator}(c_n) \cdot e\big) && = \mathsf{Indicator}(c \vee (c_1 \wedge \cdots \wedge c_n)) \cdot e\\
&\mathsf{if}\big(c,\ \ e, \mathsf{If}(c', e, e')\big) && = \mathsf{If}(c \vee c', e, e')\\
&\mathsf{if}\big(c,\ \ e, \mathsf{If}(c', e', e)\big) && = \mathsf{If}(c \vee \neg c', e, e')\\
&\mathsf{if}\big(c,\ \ e, e'\big) && = \mathsf{If}(c, e, e') \qquad \text{otherwise}
\end{aligned}
$$

Fig. 5. Auxiliary functions used in Figure 4