Stochastic Lambda Calculus and Monads of Probability Distributions

Norman Ramsey **Avi Pfeffer**

Division of Engineering and Applied Sciences
Harvard University

When worlds collide

Avi: Language for probability!

When worlds collide

Avi: Language for probability!

Norman: Semantics?

When worlds collide

Avi: Language for probability!

Norman: Semantics?

Avi: Tables. . .

When worlds collide

⋮

Result of discussion:

- Language design exposes implementation
- Must know implementation to use
- No clear semantic model

Norman is unhappy

Problem we're trying to solve

Language for probabilistic modeling

- Want nice clean semantics
- Want nice efficient implementation

This paper: solution for very simple language

(Both players happy)

Along the way

Probability, monads, and efficient models

A probabilistic language

Stochastic λ-calculus

$$e ::= x \mid v \mid \lambda x.e \mid e_1\, e_2 \mid \mathrm{let}\ x = e'\ \mathrm{in}\ e$$
$$\mid \quad \mathrm{choose}\, p\ e_1\ e_2$$
$$\mid \quad (e_1, e_2) \mid e.1 \mid e.2$$
$$\mid \quad \mathrm{L}\, e_1 \mid \mathrm{R}\, e_2 \mid \mathrm{case}\ e\ e_l\ e_r$$

Term denotes (joint) probability distribution

Why should we care? Make queries

Three kinds of queries

EXPECTATION

- Basis of most decision-making
- Subsumes probability queries

SAMPLE

- Get feel of distribution (or a picture)
- “Monte Carlo” methods
- Generate test data (Claessen and Hughes 2000)

SUPPORT

- Not interesting for users
- Important for implementation
- Reduces to simple nondeterminism

Example: Modeling Grades

Courses:

```
cs152  = dist [ 0.4 : A, 0.4 : B, 0.2 : C ]
am120  = dist [ 0.1 : A, 0.2 : B, 0.4 : C,
                0.2 : D, 0.1 : F ]
hum100 = dist [ 0.9 : A, 0.1 : B ]
```

Plan of study:

```
plan = [cs152, am120, ...]
```

Probability distribution over lists of grades

Modeling Grades, Cont'd

Grade-point average:

```
plan = [cs152, am120, ...]
weight A = 15
weight B = 11
   ⋮
gpa = avg (map weight plan)
        where avg l = foldl (+) 0 l / length l
```

Number of grades below a B:

```
belowBcount = length (filter (<B) plan)
```

Best 10 grades:

```
top10 = take 10 (sort plan)
```

Querying the Model

What might my best 10 grades look like?

SAMPLE `top10`

What's my expected GPA?

EXPECTATION `gpa`

What are my chances of getting honors (GPA over 13)?

EXPECTATION `(if gpa > 13 then 1 else 0)`

List all possible transcripts

SUPPORT `plan`

How to Answer Queries I

The probability monad

- **Long known to theory community (Lawvere 1962)**
- **Little discussed among functional programmers**

No surprises, but provides clean semantics

The Probability Monad

$M\ a$	**distribution over values of type a**
`return` a	**distribution containing a with unit probability**
$d >\!\!>\!\!= k$	**conditional probability**

$$\begin{aligned} d &= P(X) \\ k(X) &= P(Y|X) \\ d >\!\!>\!\!= k &= P(Y) = \sum_X P(X) \cdot P(Y|X) \end{aligned}$$

$choose\ p\ d_1\ d_2$	**probabilistic choice**

Nice, simple semantics and operational interpretation

Connect λ-Calculus to Monad

Terms denote distributions over values

Environments bind identifiers to values

Functions map values to distributions

$$
\begin{aligned}
\mathcal{P}[\![x]\!]\rho &= \mathtt{return}\ (\rho\, x) \\
\mathcal{P}[\![v]\!]\rho &= \mathtt{return}\ v \\
\mathcal{P}[\![\lambda x.e]\!]\rho &= \mathtt{return}\ (\lambda v.\mathcal{P}[\![e]\!]\rho\{x \mapsto v\}) \\
\mathcal{P}[\![\mathrm{let}\ x = e'\ \mathrm{in}\ e]\!]\rho &= \mathcal{P}[\![e']\!]\rho \gg\!= \lambda v.\mathcal{P}[\![e]\!]\rho\{x \mapsto v\} \\
\mathcal{P}[\![e_1\ e_2]\!]\rho &= \mathcal{P}[\![e_1]\!]\rho \gg\!= \lambda v_1.\mathcal{P}[\![e_2]\!]\rho \gg\!= \lambda v_2.v_1 v_2 \\
\mathcal{P}[\![\mathrm{choose}\, p\ e_1\ e_2]\!]\rho &= \mathit{choose}\ p\ (\mathcal{P}[\![e_1]\!]\rho)\ (\mathcal{P}[\![e_2]\!]\rho)
\end{aligned}
$$

Typical call-by-value with imperative features

Monadic Queries Are Simple

Example

```
EXPECTATION :: (a -> Double) -> M a -> Double
expectation h (return x) = h x
expectation h (d >>= k)  = expectation g d
         where g x = expectation h (k x)
expectation h (choose p d d') =
     p * expectation h d +
       (1-p) * expectation h d'
```

Any satisfying implementation computes expectation

Leads to simple code (12 lines of Haskell)

SAMPLE and SUPPORT also easy

What Have We Gained?

Monadic interpreter for stochastic λ-calculus

- **Clear semantics**
- **Simple implementation**
- **For SAMPLE and SUPPORT, efficient**
- **For EXPECTATION, trouble**

The Trouble With Expectation

Terms denote "joint" distributions over product spaces

$$\begin{array}{rcrcl}(cs152, am120, hum110) & \rightarrow & (A,B,A) & : & \mathbf{0.072} \\ & & (A,C,A) & : & \mathbf{0.144} \\ & & (B,A,A) & : & \mathbf{0.036} \\ & & \vdots & & \end{array}$$

Computing `expectation h d`

- No intensional properties; can only apply `h`
- Must apply `h` to each tuple in product space
- Exponential cost in dimension of space

Trouble and More Trouble

Look at this query:

```
EXP gpa [cs152, am120, hum100] =
   (EXP gpa [cs152] + EXP gpa [am120] +
    EXP gpa [am120]) / 3
```

Linear cost in dimensions of product space (number of variables)

Relies on "conditional independence" (very common)

A well-developed technology: **variable elimination**

Our contribution

Measure terms

- Clean language
- Clean semantics (consistent with monad)
- Can exploit decomposable queries
  (connect to tables, variable elimination)

Measure Terms

Denote measure over product space of named vars

Built from formulas, sums, products

$$t ::= \langle \phi : w \rangle \mid t_1 \times t_2 \mid t_1 + t_2 \mid \sum_{x::X} t$$

Formula ϕ assigns weights to satisfying states

Example term:

$$t = \langle cs152 = A \wedge am120 = A \wedge hum100 = B : 0.004 \rangle$$

Use $\sum$ to express variable elimination

Semantics of Measure Terms

If ρ is record binding values to variables,
$\overline{\mathcal{T}[\![t]\!]}(\rho)$ is weight assigned to that record

$$\overline{\mathcal{T}[\![\langle\phi : w\rangle]\!]}(\rho) = w \cdot sat(\phi, \rho)$$

$$sat(\phi,\rho) = \begin{cases} \textbf{1, if formula } \phi \textbf{ satisfied by assignment } \rho \\ \textbf{0, otherwise} \end{cases}$$

$$\begin{aligned}
\overline{\mathcal{T}[\![t_1 \times t_2]\!]}(\rho) &= \overline{\mathcal{T}[\![t_1]\!]}(\rho) \cdot \overline{\mathcal{T}[\![t_2]\!]}(\rho) \\
\overline{\mathcal{T}[\![t_1 + t_2]\!]}(\rho) &= \overline{\mathcal{T}[\![t_1]\!]}(\rho) + \overline{\mathcal{T}[\![t_2]\!]}(\rho) \\
\overline{\mathcal{T}[\![\textstyle\sum_{y::Y} t]\!]}(\rho) &= \textstyle\sum_{v \in Y} \overline{\mathcal{T}[\![t]\!]}(\rho\{y \mapsto v\})
\end{aligned}$$

$\sum$ "eliminates" y by summing over possible values

Clear cost model

Algebraic Laws

Operators $+$, $\times$ associative, commutative, distributive

OK to rename x in $\sum_{x::X} t$

Interchange order of sums

$\sum_{x::X}(t_1 \times t_2) = t_1 \times (\sum_{x::X} t_2)$**, provided x is not free in t_1**

Now we can optimize!

Optimizing with Algebraic Laws

Keys to fast expectation

Provided x_1 is not free in t_2 and x_2 not free in t_1,

$$\sum_{x_1::X_1} \sum_{x_2::X_2} t_1 \times t_2 = \left(\sum_{x_1::X_1} t_1 \right) \times \left(\sum_{x_2::X_2} t_2 \right)$$

Exponential on left, linear on right

Given large term, best rewrite is NP-hard

- **Use AI technique: variable elimination**

Wrapping up

Mapping $\mathcal{E}$ from stochastic λ-calculus to measure term with free variable $*$

Equivalence of two translations (a bit technical):

$$\overline{\mathcal{M}(\mathcal{P}[\![e]\!]\rho)}(v) = \overline{\mathcal{T}(\mathcal{E}[\![e]\!]\rho)}(\rho\{* \mapsto v\})$$

How to Answer Queries II

Provided

$$h(v) = w_1 \cdot sat(\phi_1, \{* \mapsto v\}) + \cdots + w_n \cdot sat(\phi_n, \{* \mapsto v\})$$

then compute EXPECTATION $h\ t$ **by**

1. **Build measure term**

$$ht = \sum_{*} (\langle \phi_1 : w_1 \rangle + \cdots + \langle \phi_n : w_n \rangle) \times t$$

2. **Optimize** ht **(variable elimination)**

3. **Evaluate** $\overline{\mathcal{T}[\![ht]\!]}(\{\})$

Related work

Lots from theory community (e.g., LICS)

- **Most accessible: Claire Jones's 1990 PhD thesis**

In functional-programming community

- **Sampling monad for testing (Claessen and Hughes 2000)**

What do we have? Where are we going?

Baby step toward

well-founded language for probabilistic modeling
with efficient queries

Still to come:

- **Bayesian inference**
- **Explicit types**
- **Lazy evaluation**
- **Infinite data structures (use Scott topologies)**
- **Memoizing lazy computations**
- **Optimization for measure terms**

Measure Term Examples

Simple term: $t = \langle x = A \wedge y = A \wedge z = B : 0.004 \rangle$

Write tables using $+$:

$$
\begin{aligned}
t \quad = \quad & \langle x = A \wedge y = A \wedge z = A : 0.036 \rangle + \\
& \langle x = A \wedge y = A \wedge z = B : 0.004 \rangle + \\
& \langle x = A \wedge y = B \wedge z = A : 0.072 \rangle + \\
& \langle x = A \wedge y = B \wedge z = B : 0.008 \rangle + \\
& \qquad \vdots
\end{aligned}
$$

GPA Example

$$\begin{aligned}
t &= \big(\langle cs152 = A : 0.4\rangle + \langle cs152 = B : 0.4\rangle + \cdots\big) \times \\
&\quad \big(\langle am120 = A : 0.1\rangle + \langle am120 = B : 0.2\rangle + \cdots\big) \times \\
&\quad \big(\langle hum100 = A : 0.9\rangle + \langle hum100 = B : 0.1\rangle\big) \\
&= t_{cs152} \times t_{am120} \times t_{hum100} \\
h(\rho) &= \tfrac{15}{3} \cdot sat(cs152 = A, \rho) + \tfrac{11}{3} \cdot sat(cs152 = B, \rho) + \cdots \\
&= h_{cs152} + h_{am120} + h_{hum100} \\
\sum_{*}(h \times t) &= \sum_{cs152} \sum_{am120} \sum_{hum100} (h_{cs152} + h_{am120} + h_{hum100}) \times t_{cs152} \times t_{am120} \times t_{hum100} \\
&= \Big(\sum_{cs152} h_{cs152} \times t_{cs152}\Big) \times \Big(\sum_{am120} t_{am120}\Big) \times \Big(\sum_{hum100} t_{hum100}\Big) \times + \cdots \\
&= \Big(\sum_{cs152} h_{cs152} \times t_{cs152}\Big) + \Big(\sum_{am120} h_{am120} \times t_{am120}\Big) + \cdots \\
\overline{\mathcal{T}[\![\textstyle\sum_{*}(h \times t)]\!]\{\}} &= \tfrac{11.8}{3} + \tfrac{7}{3} + \tfrac{14.6}{3} \approx 11.13
\end{aligned}$$