Fabular: Regression Formulas as Probabilistic Programming

Johannes Borgström
Uppsala University
Sweden

Andrew D. Gordon
Microsoft Research and
University of Edinburgh
UK

Long Ouyang
Stanford University
USA

Claudio Russo
Microsoft Research
UK

Adam Ścibior
University of Cambridge and
MPI Tübingen
Germany

Marcin Szymczak
University of Edinburgh
UK

Abstract

Regression formulas are a domain-specific language adopted by several R packages for describing an important and useful class of statistical models: hierarchical linear regressions. Formulas are succinct, expressive, and clearly popular, so are they a useful addition to probabilistic programming languages? And what do they mean? We propose a core calculus of hierarchical linear regression, in which regression coefficients are themselves defined by nested regressions (unlike in R). We explain how our calculus captures the essence of the formula DSL found in R. We describe the design and implementation of Fabular, a version of the Tabular schema-driven probabilistic programming language, enriched with formulas based on our regression calculus. To the best of our knowledge, this is the first formal description of the core ideas of R's formula notation, the first development of a calculus of regression formulas, and the first demonstration of the benefits of composing regression formulas and latent variables in a probabilistic programming language.

Categories and Subject Descriptors D.3.2 [*Programming Languages*]: Language Classifications—Specialized application languages; I.2.6 [*Artificial Intelligence*]: Learning—Parameter Learning

Keywords Bayesian inference; linear regression; probabilistic programming; relational data; hierarchical models

1. Introduction

Our goal is to embrace and extend R's hugely popular regression formulas to get better probabilistic programming languages.

1.1 Background: R's Regression Formulas

The R statistical programming language allows notation of the form $y \sim x$ to express linear regression models. If x_i, y_i are the data in row i of a table, this model expresses that each $y_i = \alpha + \beta x_i + e_i$ where e_i is an error term. Given this data and model, the *regression task* is to learn the global parameters α and β, the intercept and the slope of the line, by statistical inference.

While this example is an elementary univariate regression, the domain-specific languages of R formulas, as implemented by several different inference packages, support a wide range of classes of regressions (including multivariate, hierarchical, and generalized). The notation anonymises the parameters, such as α and β, has useful defaults, such as including the intercept and error terms automatically, and hence is extremely succinct.

Still, the published descriptions of R formulas are informal and non-compositional. If we are to transplant R formulas to other languages, the first problem is to obtain precise syntax and semantics.

1.2 Background: Probabilistic Programming

A system for probabilistic programming (Goodman 2013; Gordon et al. 2014b) asks the user to provide a probabilistic model as a piece of code, and provides a compiler to generate efficient code for statistical inference. Following the earliest system BUGS (Gilks et al. 1994; Lunn et al. 2013), there are many systems, including BLOG (Milch et al. 2007), Infer.NET (Minka et al. 2009), Church (Goodman et al. 2008), Figaro (Pfeffer 2009), HANSEI (Kiselyov and Shan 2009), Fun (Borgström et al. 2013), Stan (Stan Development Team 2014a), R2 (Nori et al. 2014), Anglican (Wood et al. 2014), Probabilistic C (Paige and Wood 2014), Venture (Mansinghka et al. 2014), and Wolfe (Riedel et al. 2014).

From the start, BUGS, Stan, and other languages have been applied to hierarchical models, but written as explicit nested loops over the data. To the best of our knowledge, no previous probabilistic programming language has adopted R's formula notation.

1.3 Part 1: Regression Calculus for Hierarchical Models

In their classic textbook, Gelman and Hill (2007) define a hierarchical/multilevel model to be "a regression (a linear or generalized linear model) in which the parameters—the regression coefficients—are given a probability model." Their textbook uses R formulas for simple regressions, but since there is no R notation for defining priors on coefficients or for directly describing hierarchical models, Gelman and Hill use probabilistic programs (in BUGS) when describing hierarchical models with priors.

Calculus of Hierarchical Regression The purpose of our regression calculus is to be a precise notation for hierarchical models with explicit priors for coefficients, and with default choices of priors to retain the succinctness of R formulas. Our calculus is inspired by R formulas and translates to probabilistic programs (in Fun, but

easily adapted to BUGS). A unique feature in our calculus is the *coefficient regression* $v\{\alpha \sim r\}$, which introduces a coefficient α together with its nested probability model r, directly corresponding to Gelman and Hill's definition of a hierarchical model.

Section 2 introduces the syntax and informal semantics of our regression calculus via a series of examples. Section 4 completes the exposition by explaining more complex examples.

Section 3 presents the standard Bayesian interpretation of regression via our calculus. We recall Fun as a syntax for interpreting regressions as typed probabilistic programs, themselves formalised in measure theory. Theorem 1 (Type Preservation) guarantees that each well-typed regression maps to a well-typed Fun expression, and hence is interpreted as a measure over the measurable space for its type. Hence, for any well-typed regression $y \sim r$ we define the *prior distribution* on its parameters and the output column of data for y; and conditioned on observed data V_y possibly with missing values, we define the *posterior distribution* on its parameters and output, which yields predictions for the missing values.

Our calculus has new features beyond R's regression formulas:

(1) The coefficient construct $v\{\alpha \sim r\}$ is a nested syntax for hierarchical linear models. R has no nested syntax for models.

(2) We can set priors for coefficients such as slopes, intercepts, or the precision of error terms.

(3) Input and output data and parameters are all typed.

Flattening Multilevel Formulas to Single Level A hierarchical regression such as $(1\{\alpha \sim r_\alpha\} \mid s) + x\{\beta \sim r_\beta\}$ includes subexpressions r_α and r_β that model the parameters α and β (here, the $|s$ syntax after $\{\alpha \sim r_\alpha\}$ indicates that we will have one α for every value of the categorical variable s). The regressions r_α and r_β may themselves include nested coefficients on group-level input data. In Section 5 we recall Gelman and Hill's discussion of how a hierarchical regression may be re-arranged so there are no nested coefficients, and by accessing group-level data from the top-level. Theorem 2 establishes that any well-typed regression has an equivalent single-level counterpart. Still, the advantage of our calculus over flattened formulas in R is that hierarchical syntax better reflects the intended structure of the model.

Explaining R's Regression Formulas We developed our calculus to be a core language to explain the regression formulas of R. Section 6 describes a semantics for the formula dialects implemented by `lm`, `lmer`, and `blmer` by mapping to the regression calculus.

1.4 Part 2: Fabular = Tabular + Regression Calculus

Tabular (Gordon et al. 2014a, 2015) is a schema-driven probabilistic programming language embedded in a spreadsheet, with inference by Infer.NET (Minka et al. 2009). In Section 7, we extend Tabular with columns defined by regressions from the regression calculus. Hence, we can express hierarchical models. In addition, since Tabular, like most probabilistic programming languages, supports latent variables defined by a model, we can use formulas to express models based on latent variables, such as clustering or ranking models. Moreover, we adopt a *vectorized interpretation* of regressions, where coefficients and outputs are vectors; hence, we obtain a formula notation for vector-based models. We develop in detail the bilinear recommender model Matchbox (Stern et al. 2009). We report a list of Fabular models we have running within the spreadsheet environment.

Formulas in Fabular have all the power of the regression calculus, and go beyond R in allowing:

(1) Use of latent variables, either continuous (such as abilities) or discrete (such as mixture components for clustering).

(2) Vectorized interpretation for examples such as Matchbox.

Section 8 concludes the paper. Technical report (Borgström et al. 2015) contains more details, definitions and proofs.

1.5 Contributions of the Paper

We propose the first formal calculus of regressions, with a unique recursive syntax for hierarchical models (based on the coefficient construct $v\{\alpha \sim r\}$), and a rigorous typed semantics. We develop a semantic equivalence for regressions, which we apply to transform multilevel regressions to equivalent single-level regressions. We explain the essence of the popular formula notation in R's `lm`, `lmer`, and `blmer` by converting formulas to terms of the regression calculus. To our knowledge, this is the first formal description of the core ideas of R's formula notation. We design and implement Fabular, a version of the Tabular schema-driven probabilistic programming language that is enriched with formulas from our regression calculus.

1.6 Related Work

We discussed probabilistic programming systems in Section 1.2. Morandat et al. (2012) conduct a careful analysis of the design of the R programming language, but do not consider its formula notation for regression. There are informal descriptions of R formulas such as (Hahn) and in the documentation for R packages. For example, Bates et al. (2014) provide a careful description of the semantics of `lmer` formulas in terms of matrix representations and algorithms. In addition, they detail instructive examples of a variety of `lmer` formulas but do not provide a grammar for this language or discuss the precise semantics of the syntax.

2. A Core Calculus of Regression, by Example

We give the syntax and informal semantics of the regression calculus, together with a series of examples. The formal typing rules and semantics are in Section 3.

2.1 Syntax and Informal Semantics

The types of the calculus are **real**, bounded naturals $\mathbf{mod}(n)$ for $n \geq 0$, and sized array types. Let s range over real constants.

Variables, Naming Conventions, and Types:

$T, U ::= \mathbf{real} \mid \mathbf{mod}(n) \mid T[n]$	type ($n \geq 0$)
x, y (continuous **real**) c, d (categorical $\mathbf{mod}(n)$) α, β, π (parameter)	
$\Gamma ::= x_1 : T_1, \ldots, x_n : T_n \quad x_i$ distinct	type environment

A core idea of the calculus is that expressions denote probability distributions over multidimensional arrays of data, referred to as cubes. A cube may be a column of predicted data, a single parameter (a zero-dimensional array), an array or a doubly-indexed array of parameters. (Higher dimensions are possible.)

Dimensions and Cube-Expressions:

Let a *dimension*, $\vec{e}$ or $\vec{f}$, be a finite list of natural numbers.
Let a *cube-expression with dimension* $\vec{e} = [e_1; \ldots; e_n]$ be a phrase that denotes a multi-dimensional array of some type $T[e_n] \ldots [e_1]$.
An *index* for $\vec{e}$ is a list $[i_1; \ldots; i_n]$ with $0 \leq i_j < e_j$ for each j.

A *predictor* v is a (deterministic) cube-expression made up of constants, variables, interactions, and paths.

Syntax of Predictors:

$u, v ::=$	predictor
s	scalar (common cases are 1 and 0)
x	variable (categorical or continuous)
$u : v$	interaction (multiplication)
$(u_1, \ldots, u_n).v$	path

A *scalar* s returns the cube with s at each index. A *variable* x returns the cube denoted by x. An *interaction* $u : v$ is the pairwise multiplication of u and v; it returns the cube with $v[\vec{i}] \times u[\vec{i}]$ at each index $\vec{i}$. Finally, a *path* $(u_1, \ldots, u_n).v$ computes an intermediate $[f_1; \ldots; f_n]$-cube for v, computes cubes $u_1, \ldots, u_n$ containing indexes for dimensions $f_1, \ldots, f_n$, and returns the cube obtained by applying the indexes from $u_1, \ldots, u_n$ to v. For instance, the form $().v$ allows a []-dimensional cube v (that is, a scalar) to be mapped to a cube of arbitrary dimension. We write $u_1.v$ for $(u_1).v$.

A *regression* r is a probabilistic cube-expression that returns a tuple of static parameters alongside a cube of outputs.

Syntax of Regressions:

$r ::=$	regression
$D(v_1, \ldots, v_n)$	noise with distribution D
$v\{\alpha \sim r\}$	predictor with coefficient
$r + r'$	sum
$r \mid v$	grouping
$(v\alpha)r$	restriction (scope of α is r)

A *noise* term $D(v_1, \ldots, v_n)$ returns a cube with an independent random draw from the distribution $D(v_1[\vec{i}], \ldots, v_n[\vec{i}])$ at each index $[\vec{i}]$. We assume the following families D of distributions.

Distributions: $D : (y_1 : U_1, \ldots, y_n : U_n) \to T$

$\mathsf{Dirac} : (\mathsf{point} : T) \to T$
$\mathsf{Gaussian} : (\mathsf{mean} : \mathbf{real}, \mathsf{variance} : \mathbf{real}) \to \mathbf{real}$
$\mathsf{Gamma} : (\mathsf{shape} : \mathbf{real}, \mathsf{rate} : \mathbf{real}) \to \mathbf{real}$

As indicated by the type signatures, the basic parameterization of a Gaussian is in terms of the mean and variance, and for a Gamma in terms of shape and scale. We write $\mathsf{Gaussian}(m, s^2)$ for the normal distribution with mean m and standard deviation s; its variance is s^2 and its precision is $1/s^2$. We allow some inverted parameterizations such as $\mathsf{Gaussian}(u, 1/v)$ for a Gaussian with mean and precision (the inverse of variance) given by u and v, and $\mathsf{Gamma}(u, 1/v)$ for a Gamma distribution with shape and rate (the inverse of scale) given by u and v. The following is a useful special case of noise: the distribution $\mathsf{Dirac}(v)$ has all its mass on v.

Derived Form of Regression:

$\delta v \triangleq \mathsf{Dirac}(v)$	deterministic case of noise: exactly v

In the simple case, a *predictor with coefficient* $v\{\alpha \sim r\}$ defines parameter α by the []-dimensional cube of r (that is, a scalar), and returns the parameters of r together with α, alongside a cube of the same dimension as v, with each component of v multiplied by α. A more complex case arises in hierarchical models, where α denotes not a single scalar coefficient, but a whole array or even multi-dimensional array of coefficients.

A *sum* $r_1 + r_2$ returns the concatenation of the parameters of r_1 and r_2 alongside the pairwise sum of their cubes.

A *grouping* $r \mid v$ introduces a hierarchical model; in the simple case, where r itself contains no grouping and v is a cube of indexes of bounded type $\mathbf{mod}(f)$, the regression $r \mid v$ is the same as r except it generates arrays of parameters of dimension $[f]$, and uses v to choose which parameter to select. In the more complex case, the parameters form an arbitrary cube.

A *restriction* $(v\alpha)r$ is the same as r, except that the parameter α returned by r is hidden.

We write $\mathrm{fv}(v)$ and $\mathrm{fv}(r)$ for the sets of variables free in v and r. We identify phrases of syntax up to alpha-conversion, the consistent renaming of bound variables. We write $\{\phi/x\}$ for syntactic substitution of phrase ϕ for variable x, avoiding capture of bound variables. For example, $(v\alpha)r = (v\alpha')(r\{\alpha'/\alpha\})$ if $\alpha' \notin \mathrm{fv}(r)$.

Let the *domain* of regression r be the list of names of parameters defined by r: we define $\mathrm{dom}(r)$ below. We write lists as $[x_1; \ldots; x_n]$ or $[x_i \ ^{i \in I}]$ for ordered index set I, and @ is list concatenation. If $\vec{\alpha} = [\alpha_i \ ^{i \in I}]$ and $j \in I$ then $\vec{\alpha} \setminus \alpha_j = [\alpha_i \ ^{i \in I \setminus j}]$.

Domain of a Regression: $\mathrm{dom}(r)$

$\mathrm{dom}(D(v_1, \ldots, v_n)) = []$
$\mathrm{dom}(v\{\alpha \sim r\}) = \mathrm{dom}(r)@[\alpha]$
$\mathrm{dom}(r_1 + r_2) = \mathrm{dom}(r_1)@\mathrm{dom}(r_2)$
$\mathrm{dom}(r \mid v) = \mathrm{dom}(r)$
$\mathrm{dom}((v\alpha)r) = \mathrm{dom}(r) \setminus \alpha$

2.2 Setting: Predicting Students' Grades

To introduce the calculus, we consider a series of example regressions for a dataset corresponding to the opening example of Gelman and Hill (2007). The dataset consists of tables of schools and students, containing $\underline{schools}$ and $\underline{students}$ rows. (Throughout we assume that the rows in a table t of size $\underline{t}$ have primary keys numbered $0, \ldots, \underline{t} - 1$, and hence we treat a table as a set of arrays of the same size.) Each student i has a property $x[i]$ (such as a pre-test score) and a school $s[i]$, while each school j has a group-level property $u[j]$ (such as average parents' incomes). We refer to the data via variables in the type environment Γ given below.

$$\begin{aligned} \Gamma \quad = \quad & u : \mathbf{real}[\underline{schools}], \\ & s : \mathbf{mod}(\underline{schools})[\underline{students}], \\ & x : \mathbf{real}[\underline{students}] \end{aligned}$$

Moreover we have a column $y = V_y$ of type $\mathbf{real}[\underline{students}]$ of test-grades, possibly with missing values. We consider a series of regressions that model y, that is, they return a cube of dimension $[\underline{students}]$. Each regression defines a joint distribution over its parameters and its output column y. If we condition this prior distribution on the observed data V_y, we obtain predictions for the missing test scores, and a posterior distribution for the model's parameters.

The task defined by a regression r plus input data matching Γ and observed output column V_y is to compute (approximations of) these conditional distributions. (We formalize in Section 3.5.)

We now consider a series of regressions for this data.

2.3 Pure Intercept: $r_1 = 1\{\alpha \sim \mathsf{Gaussian}(0, {s_{large}}^2)\}$

Our first regression r_1 is a flat baseline set by a parameter α with an uninformative prior, that is, a very wide Gaussian distribution. Our semantics for r_1 is a probabilistic program: the Fun expression shown below.

$$\begin{aligned} & \mathbf{let}\ \alpha = \mathsf{Gaussian}(0, {s_{large}}^2)\ \mathbf{in} \\ & (\alpha, [\mathbf{for}\ z < \underline{students} \to\ 1 \times \alpha]) \end{aligned}$$

(Fun is a probabilistic dialect of ML. We introduce its formal syntax in Section 3.1. A for-loop expression $[\mathbf{for}\ x < n \to F]$ produces an array $[F\{0/x\}, \ldots, F\{n-1/x\}]$.)

The expression defines α by a draw from a Gaussian with a large standard deviation s_{large}, and returns α alongside an array $y = [\mathbf{for}\ z < \underline{students} \to\ 1 \times \alpha]$ that sets each entry to α. A plot of each input $x[i]$ versus $y[i]$ for each student i is a flat line that intercepts the Y-axis at $y = \alpha$, so we refer to α as the intercept.

(In practice, choosing s_{large} is a balance between being α biased toward small numbers, and being so large as to trigger overflows. The subject of configuring priors in detail is a statistical question beyond the scope of this paper.)

In general, a regression $v\{\alpha \sim \mathsf{Gaussian}(0, {s_{large}}^2)\}$ chooses an uninformative prior for a coefficient α for a predictor v. This is a common pattern, so we allow the following abbreviations.

$$\begin{aligned} v\{\alpha\} \quad & \triangleq \quad v\{\alpha \sim \mathsf{Gaussian}(0, {s_{large}}^2\} \\ v \quad & \triangleq \quad (v\alpha)v\{\alpha\} \quad \text{for } \alpha \notin \mathrm{fv}(v) \end{aligned}$$

For instance, $1\{\alpha\}$ is the same as r above, while 1 on its own is the same as r except the parameter α is hidden.

2.4 Pure Noise: $r_2 = ?$

A model such as r_1 does not fit data unless all the points fall exactly on the intercept; the model allows the intercept to be learnt, but allows no per-point variation from the intercept. In practice, all data is noisy in that there is deviation from the line and so we need to include noise, also known as an error term. The regression written $r_2 = ?$ is a pure noise model. Its semantics is the following.

$$\begin{array}{l}\textsf{let } \pi = \mathsf{Gamma}(1, 1/\lambda_{large}) \textsf{ in} \\ ((), [\textsf{for } z < \underline{students} \rightarrow \mathsf{Gaussian}(0, 1/\pi)])\end{array}$$

Each item in the output array is a draw from $\mathsf{Gaussian}(0, 1/\pi)$, a zero-mean Gaussian with precision π. The smaller the precision the greater the variance of the noise. The precision π is drawn from distribution $\mathsf{Gamma}(1, 1/\lambda_{large})$ with a small rate parameter $1/\lambda_{large}$ to achieve a non-informative prior on the precision. The effect is that the precision of the noise is determined by the observed data.

The syntax ? is not primitive in the calculus but is derived from other constructs as follows.

$$\begin{array}{rcl} ?\{\pi \sim r\} & \triangleq & 0\{\pi \sim r\} + \mathsf{Gaussian}(0, 1/().\pi) \\ ? & \triangleq & (\nu\pi)?\{\pi \sim \mathsf{Gamma}(1, 1/\lambda_{large})\} \end{array}$$

The coefficient $0\{\pi \sim r\}$ is a coding trick that defines the parameter π by r, but makes no contribution to the output column. The literal semantics of ? is the following, though the term $0 \times \pi$ may be cancelled out.

$$\begin{array}{l}\textsf{let } \pi = \mathsf{Gamma}(1, 1/\lambda_{large}) \textsf{ in} \\ ((), [\textsf{for } z < \underline{students} \rightarrow 0 \times \pi + \mathsf{Gaussian}(0, 1/\pi)])\end{array}$$

2.5 Intercept (with Noise): $r_3 = 1\{\alpha\} + ?$

Combining an intercept and noise term yields the following model, which learns the intercept α while allowing for noise.

$$\begin{array}{l}\textsf{let } \alpha = \mathsf{Gaussian}(0, {s_{large}}^2) \textsf{ in} \\ \textsf{let } \pi = \mathsf{Gamma}(1, 1/\lambda_{large}) \textsf{ in} \\ ((\alpha), [\textsf{for } z < \underline{students} \rightarrow \alpha + \mathsf{Gaussian}(0, 1/\pi)])\end{array}$$

2.6 Slope and Intercept (with Noise): $r_4 = 1\{\alpha\} + x\{\beta\} + ?$

By including a slope $x\{\beta\}$, we obtain a regression equivalent to the R formula $y \sim x$ from Section 1.1, except that our notation includes priors on the parameters.

$$\begin{array}{l}\textsf{let } \alpha = \mathsf{Gaussian}(0, {s_{large}}^2) \textsf{ in} \\ \textsf{let } \beta = \mathsf{Gaussian}(0, {s_{large}}^2) \textsf{ in} \\ \textsf{let } \pi = \mathsf{Gamma}(1, 1/\lambda_{large}) \textsf{ in} \\ ((\alpha, \beta), [\textsf{for } z < \underline{students} \rightarrow \\ \qquad \alpha + x[z] \times \beta + \mathsf{Gaussian}(0, 1/\pi)])\end{array}$$

2.7 Varying Intercept per School: $r_5 = (1\{\alpha\} \mid s) + ?$

The regression r_5 groups the intercept on the school s, so that we learn an array α of parameters, a baseline per school.

$$\begin{array}{l}\textsf{let } \alpha = [\textsf{for } z < \underline{schools} \rightarrow \mathsf{Gaussian}(0, {s_{large}}^2)] \textsf{ in} \\ \textsf{let } \pi = \mathsf{Gamma}(1, 1/\lambda_{large}) \textsf{ in} \\ ((\alpha), [\textsf{for } z < \underline{students} \rightarrow \alpha[s[z]] + \mathsf{Gaussian}(0, 1/\pi)])\end{array}$$

The regression $1\{\alpha\} \mid s + x\{\beta\} + ?$ is the same, but has slope β.

2.8 Hierarchical (Varying-Intercept, Fixed-Slope): r_6

To take into account the school-level data u, we construct a nested regression r_α with slope b to predict each school-level parameter α. (The model r_α is similar to r_4 but at school not student level.)

$$\begin{array}{rcl} r_\alpha & = & 1\{a\} + u\{b\} + ? \\ r_6 & = & (1\{\alpha \sim r_\alpha\} \mid s) + x\{\beta\} + ? \end{array}$$

The meaning of r_α is the following:

$$\begin{array}{l}\textsf{let } a = \mathsf{Gaussian}(0, {s_{large}}^2) \textsf{ in} \\ \textsf{let } b = \mathsf{Gaussian}(0, {s_{large}}^2) \textsf{ in} \\ \textsf{let } \pi' = \mathsf{Gamma}(1, 1/\lambda_{large}) \textsf{ in} \\ ((a, b), [\textsf{for } z < \underline{schools} \rightarrow \\ \qquad a + u[z] \times b + \mathsf{Gaussian}(0, 1/\pi')])\end{array}$$

Hence, we assemble the meaning E_6 of the whole model r_6:

$$\begin{array}{l}\textsf{let } a = \mathsf{Gaussian}(0, {s_{large}}^2) \textsf{ in} \\ \textsf{let } b = \mathsf{Gaussian}(0, {s_{large}}^2) \textsf{ in} \\ \textsf{let } \pi' = \mathsf{Gamma}(1, 1/\lambda_{large}) \textsf{ in} \\ \textsf{let } \alpha = [\textsf{for } z < \underline{schools} \rightarrow \\ \qquad a + u[z] \times b + \mathsf{Gaussian}(0, 1/\pi')] \textsf{ in} \\ \textsf{let } \beta = \mathsf{Gaussian}(0, {s_{large}}^2) \textsf{ in} \\ \textsf{let } \pi = \mathsf{Gamma}(1, 1/\lambda_{large}) \textsf{ in} \\ ((a, b, \alpha, \beta), [\textsf{for } z < \underline{students} \rightarrow \\ \qquad \alpha[s[z]] + x[z] \times \beta + \mathsf{Gaussian}(0, 1/\pi)])\end{array}$$

This is the first hierarchical model of Gelman and Hill (2007).

3. Type System and Semantics of Regression

3.1 Fun: Probabilistic Expressions (Review)

Syntax of Fun We use a version of the core calculus Fun (Borgström et al. 2013) as presented by Gordon et al. (2013) with arrays of deterministic size, but without a conditioning operation within expressions.

We assume a collection of total deterministic functions g, including arithmetic and logical operators.

Expressions of Fun:

$E, F ::=$	expression
x	variable
s	constant (real, unit, int, Boolean)
$g(E_1, \ldots, E_n)$	deterministic primitive g
$D(F_1, \ldots, F_n)$	random draw from distribution D
if E_1 **then** E_2 **else** E_3	if-then-else
$[E_1, \ldots, E_n] \mid E[F]$	array literal, lookup
$[\textsf{for } x < n \rightarrow F]$	for loop (scope of index x is F)
let $x = E$ **in** F	let (scope of x is F)
(E, F)	pair
fst(E) **snd**(E)	projections

Type system of Fun We here recall the type system of Fun without zero-probability observations (Bhat et al. 2013). The syntax of types is as in Section 2, with the addition of **unit**, **bool**, **int**, and pair types $T_1 \times T_2$. We write $\Gamma \vdash E : T$ to mean that in type environment $\Gamma = x_1 : T_1, \ldots, x_n : T_n$ (x_i distinct) expression E has type T. Let $\det(E)$ mean that E contains no occurrence of $D(\ldots)$. The typing rules for Fun are standard for a first-order functional language.

Semantics of Fun Intuitively, an expression E defines a probability distribution over its return type. For each type T, we define a measurable space $\boldsymbol{T}[\![T]\!]$; probability measures on that space formalize distributions over values of the type. A *valuation* $\rho = [x_i \mapsto V_i^{\,i \in 1..n}]$ is a map from variables to values. For each expression E and valuation ρ for its free variables, we define its semantics as $\boldsymbol{P}[\![E]\!]\rho$.

Lemma 1. *If* $\Gamma \vdash E : T$ *and* $\Gamma \vdash \rho$ *then* $\mathbf{P}[\![E]\!]\rho$ *is a probability measure on* $\mathbf{T}[\![T]\!]$.

The semantics has a corresponding notion of equivalence.

Definition 1. *Let $\Gamma \vdash E_1 \equiv E_2 : T$ if and only if both $\Gamma \vdash E_1 : T$ and $\Gamma \vdash E_2 : T$ and, for all ρ, $\Gamma \vdash \rho$ implies that $\mathbf{P}[\![E_1]\!]_\rho = \mathbf{P}[\![E_2]\!]_\rho$.*

Finally, we define notation for conditioning the distribution defined by a whole expression. (We have no operators for conditioning within the syntax of expressions.) If $\Gamma \vdash E : T_1 \times \cdots \times T_n$ and $\Gamma \vdash \rho$, and for $i \in 1..m$ we have $\varnothing \vdash V_i : U_i$ and $\det(F_i)$ and $x_1 : T_1, \ldots, x_n : T_n \vdash F_i : U_i$, we write

$$\boldsymbol{P}[\![E]\!]\rho[x_1, \ldots, x_n \mid F_1 = V_1 \wedge \cdots \wedge F_m = V_m]$$

for (a version of) the *conditional probability distribution* of $\boldsymbol{P}[\![E]\!]\rho$ given that the random variable $f(x_1, \ldots, x_n) \triangleq (F_1, \ldots, F_m)$ equals $(V_1, \ldots, V_m)$.

3.2 Typing the Regression Calculus

Let a type environment Γ be of the form $x_1 : T_1, \ldots, x_n : T_n$ where the variables x_i are distinct, and let $\mathrm{dom}(\Gamma) = \{x_1, \ldots, x_n\}$.

Judgments of the Type System:

$\Gamma; \vec{e} \vdash v : T$	predictor v yields an $\vec{e}$-cube of T
$\Gamma; \vec{e}; \vec{f} \vdash r \,!\, \Pi$	regression r yields $\vec{e}$-cube with parameter $\vec{f}$-cubes

We type-check a regression r with output dimension $\vec{e}$ and parameter dimension $\vec{f}$. The effect of the regression is to introduce parameters described by the Fun context Π.

The judgment for predictors ensures that their cubes are accessible, constructible or reachable from the ambient dimensions $\vec{e}$:

Typing Rules for Predictors:

(SCALAR) $\dfrac{\Gamma \vdash \diamond \quad s \in \mathbb{R}}{\Gamma; \vec{e} \vdash s : \mathsf{real}}$ (VAR) $\dfrac{\Gamma \vdash x : T[\vec{e}]}{\Gamma; \vec{e} \vdash x : T}$ (INTERACT) $\dfrac{\Gamma; \vec{e} \vdash u : \mathsf{real} \quad \Gamma; \vec{e} \vdash v : \mathsf{real}}{\Gamma; \vec{e} \vdash (u : v) : \mathsf{real}}$

(PATH) $\dfrac{\Gamma; \vec{e} \vdash u_i : \mathsf{mod}(f_i) \quad \forall i \in 1..n \quad \Gamma; \vec{f} \vdash v : T}{\Gamma; \vec{e} \vdash (u_1, \ldots, u_n).v : T}$

In the rule (VAR), the notation $T[\vec{e}]$ is short for the multi-dimensional array $T[e_n] \ldots [e_1]$ where $\vec{e} = [e_1; \ldots; e_n]$.

Typing Rules for Regressions:

(NOISE) $\dfrac{D : (x_1 : U_1, \ldots, x_n : U_n) \to \mathsf{real} \quad \Gamma \vdash \diamond \quad \Gamma; \vec{e} \vdash u_j : U_j \;\; \forall j \in 1..n}{\Gamma; \vec{e}; \vec{f} \vdash D(u_1, \ldots, u_n) \,!\, \varnothing}$

(COEFF) $\dfrac{\Gamma; \vec{e} \vdash v : \mathsf{real} \quad \Gamma; \vec{f}; [] \vdash r \,!\, \Pi \quad \alpha \notin \mathrm{dom}(\Gamma, \Pi)}{\Gamma; \vec{e}; \vec{f} \vdash v\{\alpha \sim r\} \,!\, (\Pi, \alpha : \mathsf{real}[\vec{f}])}$

(SUM) $\dfrac{\Gamma; \vec{e}; \vec{f} \vdash r \,!\, \Pi \quad (\Gamma, \Pi); \vec{e}; \vec{f} \vdash r' \,!\, \Pi'}{\Gamma; \vec{e}; \vec{f} \vdash r + r' \,!\, (\Pi, \Pi')}$ (GROUP) $\dfrac{\Gamma; \vec{e} \vdash v : \mathsf{mod}(f) \quad \Gamma; \vec{e}; (f :: \vec{f}) \vdash r \,!\, \Pi}{\Gamma; \vec{e}; \vec{f} \vdash r \mid v \,!\, \Pi}$

(RES) $\dfrac{\Gamma; \vec{e}; \vec{f} \vdash r \,!\, (\alpha_i : T_i)^{i \in I} \quad j \in I}{\Gamma; \vec{e}; \vec{f} \vdash (v\alpha_j)r \,!\, (\alpha_i : T_i)^{i \in I \setminus j}}$

The rules for typing regressions check all subregressions are **real**-valued in the current dimension $\vec{e}$ and parameter dimension $\vec{f}$. Rule (COEFF) changes dimension from $\vec{e}$ to $\vec{f}$, entering the nested regression; Rule (GROUP) adds a categorical dimension to $\vec{f}$. All rules additionally accumulate or drop parameters introduced by the regression or its subregressions, threading an output context Π.

For example, we can derive the following. (Recall $?\{\pi \sim r\}$ is short for $0\{\pi \sim r\} + \mathsf{Gaussian}(0, 1/().\pi)$.)

(NOISE) $\dfrac{\Gamma; [] \vdash r : \mathsf{real} \quad \pi \notin \mathrm{dom}(\Gamma)}{\Gamma; \vec{e}; [] \vdash ?\{\pi \sim r\} \,!\, (\pi : \mathsf{real})}$

3.3 Translation to Pure Fun

Translation of Predictors to Fun: $[\![v]\!]\ \vec{E} = E$

$$
\begin{aligned}
&[\![s]\!]\ \vec{E} \triangleq s \\
&[\![x]\!]\ \vec{E} \triangleq x[\vec{E}] \\
&[\![u : v]\!]\ \vec{E} \triangleq [\![u]\!]\ \vec{E} \times [\![v]\!]\ \vec{E} \\
&[\![(u_1, \ldots, u_n).v]\!]\ \vec{E} \triangleq [\![v]\!]\ [[\![u_1]\!]\ \vec{E}] \ldots\ [[\![u_n]\!]\ \vec{E}]
\end{aligned}
$$

Lemma 2. *If $\Gamma; (e_i)^{i \in I} \vdash v : T$ and $\Gamma \vdash E_i : \mathsf{mod}(e_i)$ for all $i \in I$ then $\Gamma \vdash [\![v]\!]\ (E_i)^{i \in I} : T$.*

Strictly speaking the following rules are type-directed, as they assume knowledge of the typing of r. We write $[\mathsf{for}\ \vec{z} < \vec{e}\ \to E]$ short for $[\mathsf{for}\ z_1 < e_1 \to \ldots [\mathsf{for}\ z_n < e_n \to E] \ldots]$, and $E[\vec{z}]$ for $E[z_1] \ldots [z_n]$. Also, $\alpha[\vec{F}[\vec{z}]]$ below is short for $\alpha[F_1[\vec{z}]] \ldots [F_n[\vec{z}]]$. We use a pattern-matching $\mathsf{let}\ ([x_1; \ldots; x_n], y) = E_1\ \mathsf{in}\ E_2$ derivable from **fst** and **snd**.

Translation of Regressions to Fun: $[\![r]\!]\ \vec{e}\ \vec{f}\ \vec{F} = E$

$$
\begin{aligned}
&[\![D(u_1, \ldots, u_n)]\!]\ \vec{e}\ \vec{f}\ \vec{F} \triangleq ((), [\mathsf{for}\ \vec{z} < \vec{e} \to D([\![u_1]\!]\ \vec{z}, \ldots, [\![u_n]\!]\ \vec{z})]) \\
&[\![v\{\alpha \sim r\}]\!]\ \vec{e}\ \vec{f}\ \vec{F} \triangleq \mathsf{let}\ (\mathrm{dom}(r), \alpha) = [\![r]\!]\ \vec{f}\ []\ []\ \mathsf{in} \\
&\qquad\qquad (\mathrm{dom}(r) @ [\alpha], [\mathsf{for}\ \vec{z} < \vec{e}\ \to\ [\![v]\!]\ \vec{z} \times \alpha[\vec{F}[\vec{z}]]]) \\
&[\![r + r']\!]\ \vec{e}\ \vec{f}\ \vec{F} \triangleq \mathsf{let}\ (\mathrm{dom}(r), y) = [\![r]\!]\ \vec{e}\ \vec{f}\ \vec{F}\ \mathsf{in} \\
&\qquad\qquad \mathsf{let}\ (\mathrm{dom}(r'), y') = [\![r']\!]\ \vec{e}\ \vec{f}\ \vec{F}\ \mathsf{in} \\
&\qquad\qquad (\mathrm{dom}(r) @ \mathrm{dom}(r'), [\mathsf{for}\ \vec{z} < \vec{e}\ \to\ y[\vec{z}] + y'[\vec{z}]]) \\
&[\![r \mid v]\!]\ \vec{e}\ \vec{f}\ \vec{F} \triangleq [\![r]\!]\ \vec{e}\ (f :: \vec{f})\ (F :: \vec{F}) \quad \text{where} \\
&\qquad\qquad \Gamma; \vec{e} \vdash v : \mathsf{mod}(f) \text{ and } F = [\mathsf{for}\ \vec{z} < \vec{e}\ \to\ [\![v]\!]\ \vec{z}] \\
&[\![(v\alpha)r]\!]\ \vec{e}\ \vec{f}\ \vec{F} \triangleq \mathsf{let}\ (\mathrm{dom}(r), y) = [\![r]\!]\ \vec{e}\ \vec{f}\ \vec{F}\ \mathsf{in} (\mathrm{dom}(r) \setminus \alpha, y)
\end{aligned}
$$

If $\Pi = \alpha_1 : T_1, \ldots, \alpha_n : T_n$ is a regression calculus context, we define the Fun type: $\mathsf{tuple}(\Pi) = T_1 \times \cdots \times T_n$.

Theorem 1 (Type Preservation)**.** *If $\Gamma; \vec{e}; (f_j)^{j \in J} \vdash r \,!\, \Pi$ and $\Gamma \vdash F_j : \mathsf{mod}(f_j)[\vec{e}]$ for all $j \in J$, and $E = [\![r]\!]\ \vec{e}\ (f_j)^{j \in J}\ (F_j)^{j \in J}$, then we have $\Gamma \vdash E : \mathsf{tuple}(\Pi) \times \mathsf{real}[\vec{e}]$.*

Proof: By induction on the derivation of $\Gamma; \vec{e}; (f_j)^{j \in J} \vdash r \,!\, \Pi$. ∎

Recall Γ, r_α, r_6, E_6 from Section 2. We have $\Gamma; [\underline{schools}]; [] \vdash r_\alpha : (a : \mathsf{real}, b : \mathsf{real})$. We also have $\Gamma; [\underline{students}]; [] \vdash r_6 : \Pi$ where $\Pi = a : \mathsf{real}, b : \mathsf{real}, \alpha : \mathsf{real}[\underline{schools}],\ \beta : \mathsf{real}$. We have $E_6 \equiv [\![r_6]\!]\ [\underline{students}]\ []\ []$. Hence, by Theorem 1 (Type Preservation), $\Gamma \vdash E_6 : \mathsf{tuple}(\Pi) \times \mathsf{real}[\underline{students}]$.

3.4 Data for Regression: Column-Oriented Databases

We consider regression in the context of a column-oriented database, where the columns consist of arrays of values, and columns of the same size are grouped into tables. Let t range over table names and c range over column names. We consider a database to be a pair $DB = (\delta_{in}, \rho_{sz})$ consisting of a record of tables $\delta_{in} = [t_i \mapsto \tau_i\ ^{i \in 1..n}]$, where each table $\tau_i = [c_{i,j} \mapsto \mathsf{inst}(V_{i,j})\ ^{j \in 1..m_i}]$ is a record of columns $V_{i,j}$, together with a valuation $\rho_{sz} = [t_i \mapsto t_i\ ^{i \in 1..n}]$ holding the number of rows $t_i \in \mathbb{N}$ in each column $V_{i,j}$ of table t_i.

To name the columns in a database, we flatten it into an environment with an array-typed variable for each column. Consider an environment Γ and a valuation ρ. We say that Γ and

ρ *match DB* to mean that $\Gamma = (c_{i,j} : T_{i,j}[\rho_{sz}(t_i)])^{i\in 1..n, j\in 1..m_i}$ and $\rho = [(c_{i,j} \mapsto V_{i,j})^{i\in 1..m, j\in 1..p_i}]$ and that $\Gamma \vdash \rho$. The latter implies that the column names $c_{i,j}$ are pairwise distinct. We write $\Gamma \vdash \rho$ to mean that the values in ρ match the types in Γ, that is, $\Gamma \vdash [x_i \mapsto V_i^{i\in 1..n}]$ if and only if $\Gamma = (x_i : T_i)^{i\in 1..n}$ and $\varnothing \vdash V_i : T_i$ for all $i \in 1..n$.

For example, consider a database for our schools example:

$$\begin{aligned} DB &= (\delta_{in}, \rho_{sz}) \\ \rho_{sz} &= [schools \mapsto 20, students \mapsto 200] \\ \delta_{in} &= [schools \mapsto \tau_{schools}, students \mapsto \tau_{students}] \\ \tau_{schools} &= [u \mapsto \mathbf{inst}(V_{1,1})] \\ \tau_{students} &= [x \mapsto \mathbf{inst}(V_{2,1}), s \mapsto \mathbf{inst}(V_{2,2})] \end{aligned}$$

We have that Γ and ρ match *DB*, where Γ is as in Section 2.2 and $\rho = [u \mapsto V_{1,1}, x \mapsto V_{2,1}, s \mapsto V_{2,2}]$.

3.5 Semantics of Regression

Consider Γ and ρ that match the input database *DB*, so that $\Gamma \vdash \rho$. We wish to use the regression r_y as model for a column y on a table t of size $\underline{t}$, where y does not appear in *DB*. We then define the *prior distribution* μ of the tuple $(\mathrm{dom}(r_y), y)$ of coefficients and y as

$$\mu \triangleq \boldsymbol{P}[\![E_y]\!]\rho \text{ where } E_y = [\![r_y]\!]\ [\underline{t}]\ []\ [].$$

Lemma 3. *Suppose that* $\Gamma \vdash \rho$ *and* $\Gamma; [\underline{t}]; [] \vdash r_y\ !\ \Pi$. *Then* μ *is a probability measure on the measurable space* $\mathbf{T}[\![\mathsf{tuple}(\Pi) \times \mathbf{real}[\underline{t}]]\!]$.

Proof: By Theorem 1 (Type Preservation), $\Gamma; [\underline{t}]; [] \vdash r_y\ !\ \Pi$ implies that $\Gamma \vdash E_y : \mathsf{tuple}(\Pi) \times \mathbf{real}[\underline{t}]$ where $E_y = [\![r_y]\!]\ [\underline{t}]\ []\ []$. By Lemma 1 and inversion of typing, $\Gamma \vdash E_y : \mathsf{tuple}(\Pi) \times \mathbf{real}[\underline{t}]$ and $\Gamma \vdash \rho$ imply that $\boldsymbol{P}[\![E_y]\!]\rho$ is a probability measure on $\boldsymbol{T}[\![\mathsf{tuple}(\Pi) \times \mathbf{real}[\underline{t}]]\!]$. ■

To apply a regression, we need observations of some or all of the items in the column y predicted by r_y. Consider O a subset of the indexes of y, that is, $O \subseteq \{i \mid 0 \le i < \underline{t}\}$. Let an *O-observation on* $\underline{t}$ be an array of observed values for the indexes in O, that is, an array $V_y = [d_0, \ldots, d_{\underline{t}-1}]$ such that $d_i \in \mathbb{R}$ if $i \in O$, and otherwise $d_i = \_$ where $\_$ represents a missing value.

The *posterior distribution* of $(\mathrm{dom}(r_y), y)$ given an O-observation V_y on $\underline{t}$ is the conditional distribution

$$\mu[\vec{\alpha}, y \mid y_i = V_y[i] \text{ for all } i \in O]$$

which is μ conditioned on the outputs $y[i]$ being equal to the observed values d_i where $i \in O$. Marginalizing this distribution yields the posterior for each parameter in $\vec{\alpha}$ and the posterior prediction for each unobserved output $y[i]$ for $i \notin O$.

Semantic equivalence for regressions is given by semantic equivalence of the corresponding Fun terms.

Definition 2. *Let* $\Gamma; \vec{e}; (f_j)^{j\in J} \vdash r_1 \equiv r_2\ !\ \Pi$ *if and only if both* $\Gamma; \vec{e}; (f_j)^{j\in J} \vdash r_i\ !\ \Pi$ *for* $i \in 1..2$ *and for all* $(F_j)^{j\in J}$ *such that* $\Gamma \vdash F_j : \mathbf{mod}(f_j)[\vec{e}]$, *if* $E_i = [\![r_i]\!]\ \vec{e}\ (f_j)^{j\in J}\ (F_j)^{j\in J}$ *then* $\Gamma \vdash E_1 \equiv E_2 : \mathsf{tuple}(\Pi) \times \mathbf{real}[\vec{e}]$.

4. Examples: Radon and InstEval

We now resume the exposition of models in the regression calculus from Section 2, by explaining how our calculus captures typical multi-level models and techniques such as partial pooling and multiple grouping, as illustrated by the radon example from Gelman and Hill (2007), and the InstEval example from Bates et al. (2014).

4.1 Example of Partial Pooling: Radon

Gelman and Hill (2007) consider a specific data set that contains radiation measurements taken in houses across different counties in Minnesota. Each measurement includes the floor f (either 0 or 1, represented by a **real**) and the radon activity activity and the county c of the house. For each county, we have a background level of uranium radiation u.

$$\begin{aligned} \Gamma \quad = \quad & \mathsf{u} : \mathbf{real}[\underline{counties}], \\ & \mathsf{c} : \mathbf{mod}(\underline{counties})[\underline{houses}], \\ & \mathsf{f} : \mathbf{real}[\underline{houses}] \end{aligned}$$

We treat floor as a continuous predictor and presume that the county-level intercept depends on the uranium level in each county—in particular, this intercept is a linear function of uranium level. Our model r_7 defines radiation activity in terms of the floor the measurement was taken on (basement, first floor, ...) and the county:

$$\begin{aligned} r_\alpha &= 1\{a\} + \mathsf{u}\{b\} + ?\{\pi \sim \delta v\} \\ r_7 &= (1\{\alpha \sim r_\alpha\} \mid \mathsf{c}) + \mathsf{f}\{\beta\} + ? \\ \Pi &= a, b, \pi : \mathbf{real}, \alpha : \mathbf{real}[\underline{counties}], \beta : \mathbf{real} \end{aligned}$$

(This concrete dataset is in fact isomorphic to the hypothetical schools example. The model is isomorphic to r_6 except that we are exposing the precision parameter π for the group-level noise.)

Our purpose with this example is to discuss a critical feature: *partial pooling* of data across different groups of observations. Partial pooling is an interpolation between *no pooling*, where observations from different groups are treated completely separately, and *complete pooling*, where observations from different groups are treated identically.

For example, in the radon data set, suppose that we wish to model radon activity in a house simply as a function of the county that the house is located in. This means that $y[i] = \alpha[\mathsf{c}[i]] + \text{noise}$, where $y[i]$ is the measured radon activity in house i and $\alpha[\mathsf{c}[i]]$ is an intercept for county $\mathsf{c}[i]$, or as a regression calculus formula, $1\{\alpha \sim r_\alpha\} \mid \mathsf{c} + ?$.

In complete pooling, we set a single α for all counties, i.e,. $\forall i, j$ $\alpha[\mathsf{c}[i]] = \alpha[\mathsf{c}[j]]$, which is equivalent to leaving off the grouping $\mid \mathsf{c}$ from the regression calculus formula above. In no pooling, we independently fit the $\alpha[c]$ (in particular, we sample these values from a distribution with known parameters so that the individual $\alpha[c]$ are conditionally i.i.d.). Partial pooling is a compromise between these two extremes; we sample the $\alpha[c]$ from some distribution with uncertain parameters, e.g.,

$$1\{\alpha \sim \mathsf{Gaussian}(\mu_\alpha, \sigma_\alpha)\} \mid \mathsf{c}$$

where μ_α and σ_α are estimated from the data. Because there is uncertainty about these parameters, the individual $\alpha[c]$ are not independent; information about all counties informs the estimate for any particular county.

The package `lmer` (cf. Section 6.2) allows for concise representation of partial pooling using the syntax `activity` $\sim$ `(1|county)` but it cannot express the no- or complete pooling variants; these must be expressed using `lm`, R's method for ordinary least squares regression (cf. Section 6.1). This is slightly awkward, as it means that exploring small variations on the model for $\alpha[\mathsf{c}[i]]$ (e.g., comparing complete pooling with partial pooling) would require switching between two different R packages.

By contrast, our calculus cleanly expresses all three possibilities based on the following template:

$$\begin{aligned} r_{county} &= 1\{a\} + ?\{\eta \sim r_\eta\} \\ r_{house} &= (1\{\alpha \sim r_{county}\} \mid \mathsf{c}) + ? \end{aligned}$$

Here, r_{county} is the county-level regression, and r_{house} is the house-level regression. To get complete pooling, we set r_η to $\delta\infty$. This maximal precision for the county-level noise will result in all α having the same value across counties. To get no pooling, we set r_η to $\delta 0$. This minimal precision for the county-level noise will result in the α essentially being free parameters. To get partial pooling,

we set r_η to $\mathsf{Gamma}(1, 1/\lambda_{large})$. By placing uncertainty over the precision η, we allow information to "flow" between counties—information about one county informs estimates about others. For concreteness, here is a simplified Fun translation of the above model (with r_η left unspecified, and assuming that $\mathrm{dom}(r_\eta) = []$):

$$
\begin{array}{l}
\textbf{let } a = \mathsf{Gaussian}(0, s^2_{large}) \textbf{ in} \\
\textbf{let } ((), \eta) = [\![r_\eta]\!]\ [\,]\ [\,]\ [\,] \textbf{ in} \\
\textbf{let } \alpha = [\textbf{for } z < \underline{counties} \to a + \mathsf{Gaussian}(0, 1/\eta)] \textbf{ in} \\
\textbf{let } \pi = \mathsf{Gamma}(1, 1/\lambda_{large}) \textbf{ in} \\
((a, \eta, \alpha), [\textbf{for } z < \underline{houses} \to \alpha[\mathsf{c}[z]] + \mathsf{Gaussian}(0, 1/\pi)])
\end{array}
$$

4.2 Example of Multiple Grouping: InstEval

One formula pattern that is possible with hierarchical models is what we call *multiple grouping*: grouping on the Cartesian product of multiple variables. For example, consider an analysis of the InstEval dataset (from the lme4 package by Bates et al. (2014)) which contains ratings that ETH Zurich students gave their professors:

```
rating ~ (1|student) + (1|professor) +
         (1|department:service)
```

Here, `department` indicates the department that the course was taught in and `service` indicates whether the course was a service course taught to students outside the department. We model the rating as depending on three random effects intercepts: one that varies by student (e.g., some students may tend to give high ratings), another that varies by professor (e.g., some professors may be particularly well-liked), and another that varies by the interaction of course department with service status (e.g., some departments might put less effort into their service courses than others and thus receive lower ratings). We obtain the same behavior in our calculus by composing grouping operators:

$$
\begin{array}{rl}
\Gamma = & \mathsf{student} : \textbf{mod}(\underline{students})[\underline{ratings}], \\
& \mathsf{course} : \textbf{mod}(\underline{courses})[\underline{ratings}], \\
& \mathsf{professor} : \textbf{mod}(\underline{professors})[\underline{courses}], \\
& \mathsf{department} : \textbf{mod}(\underline{departments})[\underline{courses}], \\
& \mathsf{service} : \textbf{mod}(2)[\underline{courses}] \\
r_{rating} = & (1\{a\} \mid \mathsf{student}) + (1\{b\} \mid \mathsf{course.professor}) + \\
& ((1\{c\} \mid \mathsf{course.department}) \mid \mathsf{course.service}) + ? \\
\Pi = & a : \textbf{real}[\underline{students}], b : \textbf{real}[\underline{professors}], \\
& c : \textbf{real}[2][\underline{departments}]
\end{array}
$$

5. Reducing Multilevel to Classical Regression

5.1 Equivalent Formulations of the Radon Example

Hierarchical linear models can be written in several equivalent forms, as demonstrated in section 12.5 of Gelman and Hill (2007). The essence of such equivalence is that predictors can be placed at different levels of a regression and it can be useful both for understanding the model and for computational reasons. In this section we use the Radon model as an example to present such an equivalence and then we develop an equational theory for our regression calculus. We use it to prove that every regression can be reduced to a certain normal form.

Recall the Radon model with no or complete pooling from Section 4.1 (where we omit the known precision $\pi = 1/s$):

$$
\begin{array}{rl}
r_\alpha = & 1\{a\} + \mathsf{u}\{b\} + \mathsf{Gaussian}(0, s) \\
r_{activity} = & (1\{\alpha \sim r_\alpha\} \mid \mathsf{c}) + \mathsf{f}\{\beta\} + ? \\
\Pi = & a, b, : \textbf{real}, \alpha : \textbf{real}[\underline{counties}], \beta : \textbf{real}
\end{array}
$$

We can write it using a single formula in three equivalent forms:

$$
\begin{array}{l}
r_1 = (1\{\alpha_1 \sim 1\{a\} + \mathsf{u}\{b\} + \mathsf{Gaussian}(0, s)\} \mid \mathsf{c}) + \mathsf{f}\{\beta\} + ? \\
r_2 = (\mathsf{c.u})\{b\} + (1\{\alpha_2 \sim 1\{a\} + \mathsf{Gaussian}(0, s)\} \mid \mathsf{c}) + \mathsf{f}\{\beta\} + ? \\
r_3 = 1\{a\} + (\mathsf{c.u})\{b\} + (1\{\alpha_3 \sim \mathsf{Gaussian}(0, s)\} \mid \mathsf{c}) + \mathsf{f}\{\beta\} + ?
\end{array}
$$

The regressions are equivalent in the sense of producing the same predictions and the same posteriors for parameters a, b, β, but their α parameters are slightly different. They are related by

$$\alpha_1[c] = \alpha_2[c] + c.u \times b = \alpha_3[c] + a + c.u \times b.$$

The r_3 form is particularly interesting in that the county level contains no inner coefficients. Below we show that every multilevel regression can be put in such a form.

5.2 Every Regression Has Single-Level Counterpart

We here give an algorithm that normalizes a regression to an equivalent single-level regression. We first define specific classes of terms that help state the normal form, the algorithm, and its correctness theorem.

We use the path notation $\vec{u}.v$ as short for $(u_1 \ldots . u_n).v$. We write $\Sigma_{i=1}^n r_i$ for the regression $r_1 + \ldots + r_n$, letting $\Sigma_{i=1}^0 r_i = \delta 0$. We also write $(v\beta, \vec{\alpha})r$ for $(v\beta)(v\vec{\alpha})r$, and $(v)r$ for r.

Classes of Terms: N, P

$N ::= \Sigma_{i=1}^n D_i(u_{i1}, \ldots, u_{i\lvert D_i\rvert})$	noise
$P ::= \Sigma_{i=1}^n v_i\{\alpha_i \sim N_i\} \mid \vec{w}_i$	single-level regression

Every regression r normalizes to a single-level regression of the form $(v\vec{\alpha})(P + N)$. Here $\vec{\alpha}$ contains all restrictions in r, as well as the auxiliary coefficients that can be used to reconstruct original coefficients, as seen above in Section 5.1. The term N describes the noise. P intuitively contains two different kinds of terms: coefficients $v\{\alpha \sim N\} \mid \vec{w}$, and post-processing terms $0\{\beta \sim N\} \mid \vec{w}$ that compute the original coefficient β in terms of $\vec{\alpha}$ (which appear as part of N).

The algorithm applies constant folding, written $\mathrm{cf}(u)$, to predictors: paths ending in scalars simplify to the scalar, and interactions with the scalars 0 and 1 are simplified (0 is an absorbing element for interaction, 1 is a unit).

Normalization of regressions: $r \mid \vec{w} \Downarrow (v\vec{\alpha})(P + N)$

(NORM NOISE)

$$\frac{}{D(v_1, \ldots, v_n) \mid \vec{w} \Downarrow \delta 0 + D(v_1, \ldots, v_n)}$$

(NORM RES)

$$\frac{r \mid \vec{w} \Downarrow (v\vec{\alpha})(P + N) \qquad \beta \notin \vec{\alpha}, \vec{w}}{((v\beta)r) \mid \vec{w} \Downarrow (v\beta, \vec{\alpha})(P + N)}$$

(NORM GROUP)

$$\frac{r \mid v, \vec{w} \Downarrow (v\vec{\alpha})(P + N)}{(r \mid v) \mid \vec{w} \Downarrow (v\vec{\alpha})(P + N)}$$

(NORM PLUS)

$$\frac{\begin{array}{ll} r_1 \mid \vec{w} \Downarrow (v\vec{\alpha}_1)(P_1 + N_1) & \vec{\alpha}_1 \cap \mathrm{fv}(\vec{\alpha}_2, P_2, N_2) = \varnothing \\ r_2 \mid \vec{w} \Downarrow (v\vec{\alpha}_2)(P_2 + N_2) & \vec{\alpha}_2 \cap \mathrm{fv}(\vec{\alpha}_1, P_1, N_1) = \varnothing \end{array}}{(r_1 + r_2) \mid \vec{w} \Downarrow (v\vec{\alpha}_1, \vec{\alpha}_2)((P_1 + P_2) + (N_1 + N_2))}$$

(NORM COEFF NOISE)

$$\frac{r \mid \varepsilon \Downarrow (v\vec{\alpha})(\delta 0 + N) \qquad \vec{\alpha} \cap \mathrm{fv}(v, \alpha, \vec{w}) = \varnothing}{v\{\alpha \sim r\} \mid \vec{w} \Downarrow (v\vec{\alpha})(v\{\alpha \sim N\} \mid \vec{w} + \delta 0)}$$

(NORM COEFF)

$$\frac{\begin{array}{ll} r \mid \varepsilon \Downarrow (v\vec{\alpha})(P + N) & P = \Sigma_{i=1}^n u_i\{\beta_i \sim t_i\} \mid \vec{v}_i \qquad n > 0 \\ \beta' \cap \mathrm{fv}(\vec{\alpha}, v, \alpha, r, \vec{w}) = \varnothing & \vec{\alpha} \cap \mathrm{fv}(v, \alpha, \vec{w}) = \varnothing \\ \vec{w}.(x_1, \ldots, x_m) := \vec{w}.x_1, \ldots, \vec{w}.x_m & \\ P' = \Sigma_{i=1}^n \mathrm{cf}(v : \vec{w}.u_i)\{\beta_i \sim t_i\} \mid \vec{w}.\vec{v}_i & \\ r' = 0\{\alpha \sim \Sigma_{i=1}^n \delta\, \mathrm{cf}(u_i : \vec{v}^i.\beta_i) + \delta\beta'\} \mid \vec{w} & \end{array}}{v\{\alpha \sim r\} \mid \vec{w} \Downarrow (v\vec{\alpha}, \beta')((P' + v\{\beta' \sim N\} \mid \vec{w} + r') + \delta 0)}$$

Noise terms lose any grouping $\vec{w}$. Restrictions and groupings are simply recursed into. The normal form of a sum is the sum of the normal forms, rearranged to match the desired format. The interesting case is a predictor v whose coefficient is given by an inner regression with normal form $(v\vec{a})(P+N)$. If there are no inner coefficients (i.e., $P = \delta 0$), we simply rearrange the term to fit the format. Otherwise, we create a fresh coefficient β' for the noise term N. Each term $u_i\{\beta_i \sim t_i\} \mid \vec{v}_i$ of P is normalised to an interaction between the top-level predictor v and the inner predictor u_i, where u_i is obtained through the path $\vec{w}$. The regression formula giving the coefficient of this interaction is unchanged (t_i), though its conditions $\vec{v}_i$ now need to be obtained through the path $\vec{w}$. Finally, the term r' gives the original regression coeffient α as a sum of interactions between the predictors u_i in P and their coefficients β_i (obtained via the path $\vec{v}_i$), plus the fresh coefficient β'.

The special case treated by (NORM COEFF NOISE) ensures that the common patterns $v\{\alpha\}$ and $?\{\pi \sim N\}$ normalize to themselves, i.e., $v\{\alpha\} \mid \varepsilon \Downarrow v\{\alpha\}$ and $?\{\pi \sim N\} \mid \varepsilon \Downarrow ?\{\pi \sim N\}$.

The normal form always exists, and is unique.

Lemma 4. *For all $r, \vec{w}$ there exist $\vec{\alpha}, P, N$ such that $r \mid \vec{w} \Downarrow (v\vec{\alpha})(P+N)$. Moreover, if $r \mid \varepsilon \Downarrow (v\vec{\alpha}')(P'+N')$ then $(v\vec{\alpha})(P+N) = (v\vec{\alpha}')(P'+N')$.*

The normal form has the same type as the original regression formula, and represents the same prior probability distribution.

Theorem 2. *If $\Gamma; \vec{e}; [] \vdash r : \Pi$ and $r \mid \varepsilon \Downarrow (v\vec{\alpha})(P+N)$ then $\Gamma; \vec{e}; [] \vdash r \equiv (v\vec{\alpha})(P+N)\ !\ \Pi$.*

Proof: The proof is via a typed equational theory for regressions, which is proven sound with respect to $\equiv$ via a typed equational theory for Fun. ■

In the Radon example at the beginning of this section, we have

$$r_1 \Downarrow (v\alpha_3)(1\{a\} + c.u\{b\} + (1\{\alpha_3 \sim \mathsf{Gaussian}(0,s)\} \mid \mathsf{c}) + (0\{\alpha_1 \sim \delta a + \delta u : b + \delta\alpha_3\} \mid c) + f\{\beta\} + ?$$

$$r_2 \Downarrow (v\alpha_3)(c.u\{b\} + 1\{a\} + (1\{\alpha_3 \sim \mathsf{Gaussian}(0,s)\} \mid \mathsf{c}) + (0\{\alpha_2 \sim \delta a + \delta\alpha_3\} \mid c) + f\{\beta\} + ?$$

Thus, the normal forms of both r_1 and r_2 have the same terms as r_3, apart from one additional term from which we immediately can read off the relationship between α_3 and α_1 (resp. α_2).

When $r \Downarrow (v\vec{\alpha})(P+N)$, the single-level regression $P+N$ is a flattened form of r, and can be solved by many methods. The multilevel regression r directly expresses the modeller's intent in terms of group-level coefficients that themselves are modelled, and should be easier to read and understand as its structure follows the structure of the schema.

6. The Essence of R's Regression Formulas

Here, we show that our regression calculus explains the formula languages recognized by three R methods: `lm`, `lmer`, and `blmer`.

6.1 lm

Base R (R Core Team 2015) provides a method `lm` for fitting linear regressions. The core syntax of formulas recognised by `lm` is:[1]

lm grammar:

$R ::=$	regression
$\quad t_1 + \cdots + t_n + 0$	without intercept
$\quad t_1 + \cdots + t_n + 1$	with intercept
$t ::=$	predictor
$\quad x_1 : \cdots : x_m : c_1 : \cdots : c_n$	$m + n > 0$

Here, R is a regression, 0 (1) disables (enables) fitting an intercept, t_i are predictor terms in the regression, x_i are continuous variables, c_i are categorical variables. Observe that the t_i terms are interactions between any number of continuous or categorical variables. In addition, `lm` always assumes normally distributed noise. Any `lm` formula can be easily translated as an expression in our calculus, as such formulas require only sum and interaction terms from our syntax, as can be seen from our translation:

$$\begin{aligned} [\![t_1 + ... + t_n + 0]\!] &= [\![t_1]\!] + ... + [\![t_n]\!] + ? \\ [\![t_1 + ... + t_n + 1]\!] &= [\![t_1]\!] + ... + [\![t_n]\!] + 1\{\alpha\} + ? \\ [\![x_1 : ... : x_m : c_1 : ... : c_n]\!] &= 1 : x_1 : ... : x_m\{\alpha\} | c_1 | ... | c_n \end{aligned}$$

We independently translate all terms in the top level regression; note that 0 translates to just noise (no intercept) whereas 1 translates to an intercept along with noise. We translate interaction terms using our grouping syntax, fitting coefficients for the product of all continuous predictors (if any) conditional on all of the categorical predictors.

The following `lm` formula is discussed by Dorie (2014):

```
ravens_z ~ treatment + initial_age
```

The formula is a model for data from Whaley et al. (2003), who performed nutrition interventions on students in twelve rural Kenyan schools. Each school was randomly assigned to one of four interventions and children at those schools took cognitive assessments before, during, and after the intervention. Running `lm` with this formula will regress the standardized Raven's score (the cognitive assessment) on the treatment the child received and the initial age of the child.

6.2 lmer

The `lmer` method, provided in the lme4 package (Bates et al. 2014), extends `lm` by adding one or more *random effects*.

lmer grammar:

$s ::=$	lmer regression
$\quad r_0 + (r_1 \mid g_1) + ... + (r_n \mid g_n)$	Fixed and random effects
$g ::=$	Grouping variable
$\quad c_1 : ... : c_m$	Product of discrete predictors
$r ::=$	lm regression
$\quad t_1 + \cdots + t_n + 0$	without intercept
$\quad t_1 + \cdots + t_n + 1$	with intercept
$t ::=$	predictor
$\quad x_1 : \cdots : x_m : c_1 : \cdots : c_n$	m + n > 0

Random effects terms take the form $(\Sigma_{i=1}^{n} t_i) | g$, indicating that the effect of the predictors in $\vec{t}$ depends on the value of the grouping factor g, which is a Cartesian product of discrete predictors in the table. For example, `weight` $\sim$ (`age` + `height`|`gender`) indicates that the coefficients that relate age and height to weight (as well as an implicit intercept) vary by gender (in a partially pooled fashion, as discussed earlier). As with `lm`, it is not possible to set priors

[1] In addition, the following constructs recognised by `lm` can be seen as macros that expand to formulas in the core syntax: ^ (used for controlling the arity of interaction terms), * (x*y is shorthand for x + y + x:y), / (used to indicate nesting of variable levels within each other, e.g., c/d desugars to c + c:d), and I (used to temporarily supplant the regression meanings of +, *, and ^ with their traditional arithmetic meanings). Furthermore, formulas can include transformations of variables, e.g., log(`area`). We omit these features because they can all be captured by +, :, and appropriate preprocessing of predictors.

on the coefficient values. `lmer` formulas can be translated[2] to our calculus by extending our translation for `lm` with the following rule to handle random effects:

$$[\![x_1 : ... : x_n : c_1 : ... : c_m | d_1 : ... : d_j]\!] \\ = \quad (1 : x_1 : ... : x_n) \mid c_1 \mid \cdots \mid c_m \mid d_1 \mid \cdots \mid d_j$$

Continuing the nutrition example introduced above, one possible `lmer` formula is:

```
ravens.z ~ treatment + initial.age + (1|school)
```

Here, we fit a separate baseline for each school, expressing the belief that baseline measures of cognition may differ across schools (e.g., one school may be in a wealthier area where parents can afford after-school tutors).

It is worth noting that actually running this model in `lmer` returns a degenerate result where all schools have the *same* baseline (no variation across schools); this is because `lmer` uses maximum likelihood estimation, which can hit this boundary condition when only small amounts of data are available. This problem can be avoided by setting a priors on the fixed/random effects terms so as to avoid the boundary, but `lmer` does not support this.

6.3 blmer

The `blmer` method, provided in the blme package (Dorie 2014), uses the same syntax of regressions as `lmer`, but additionally supports setting priors on the fixed effects (Gaussian and t priors only) and random effects (arbitrary priors). To round out the nutrition example, we can express partial pooling using `blmer` like so:

```
blmer(ravens_z ~ treatment + initial_age + (1|school),
      cov.prior = gamma(2.5, 0))
```

Here, we place a Gamma$(2.5, 0)$ prior on the covariance matrix for the terms specified by $(1|\texttt{school})$. This essentially expresses the belief that the variation in baseline cognitive scores across schools should be non-zero. In our calculus, we can write the same model using the regression formula

$$\mathsf{treatment} + \mathsf{initial\_age} + (1\{\alpha \sim ?\{\pi \sim \mathsf{Gamma}(2.5, 0)\}\} \mid \mathsf{school}).$$

Note that `blmer` does not define a *language* for setting priors but rather enables this through method arguments (e.g., the `cov.prior` argument set above). By contrast, our calculus allows to set priors compactly and compositionally. To our knowledge, our regression calculus admits a superset of the models expressible in `blmer`.

7. Fabular = Tabular + Regression Formulas

Tabular (Gordon et al. 2014a, 2015) is a table-oriented probabilistic programming language embedded in Excel. A Tabular program is a *schema* that lists a sequence of named tables. In turn, each table is described by sequence of named attribute declarations. A **static** attribute declares a value that is shared amongst all rows of the table. An **inst** (instance) attribute, on the other hand, declares a column of values in that table. The definition of an attribute is a Fun expression that may refer to the value of any previously declared attribute, whether static or, for an instance level attribute, the value of any previous instance attribute (belonging to the same row). Tabular expressions may dereference attributes of other tables using a dot-notation like syntax that, for instance level columns, corresponds to array indexing. Attributes declared as **input** take their (deterministic) values from an input database (in Excel, the database is the collection of Excel tables). Attributes marked as **output** have a probabilistic definition—a Fun expression E—denoting a random variable. If an output is present in the input database, then its deterministic value is used to condition the static or instance level value of the corresponding random variable. If an output is missing (that is, null) in the database, then its output is given as the (marginal) distribution of its definition. Output attributes may be missing from all, some or none of the rows in a column. If the attribute name is not present in the database, then the attribute is itself a latent variable or table-sized collection of latent variables. Finally, **local** attributes are similar to **output** attributes but statically scoped to the current table. Unlike inputs and outputs, local attributes cannot be referenced from other tables (even via links).

A Tabular schema defines a generative model of the database. Conditioning the model on the database allow us to infer the distributions of missing values and latent columns. Tabular is compiled to a lower level Infer.NET model that performs message-passing inference, yielding approximations to the marginal distributions of all missing values.

In this section, we extend Tabular with linguistic support for regressions. In Tabular, an attribute defined by a regression is expanded into a sequence of attributes defining the static parameters of the regression and an eponymous body defined as a sum of products and noise. Schemas may contain multiple regressions. In the full Tabular language, regressions may also occur within Tabular functions, supporting useful abstraction.

7.1 Core Tabular (Review)

Databases Tabular acts on databases of the following form, a slight generalization of the databases from Section 3.1 to include singleton values (**static**(V)) for static columns and the null value (_) denoting a missing value of any type.

Databases, Tables, Attributes, and Values:

$\delta_{in} ::= [t_i \mapsto \tau_i \ ^{i \in 1..n}]$	whole database
$\tau ::= [c_j \mapsto a_j \ ^{j \in 1..m}]$	table in database
$a ::= \ell(V)$	attribute value: V with level ℓ
$V ::= \_ \mid s \mid [V_0, \ldots, V_{n-1}]$	nullable value
$\ell, pc ::= \mathbf{static} \mid \mathbf{inst}$	level ($\mathbf{static} < \mathbf{inst}$)

Schemas We use the Fun types T from Section 3.1 in Tabular. We write $\mathbf{link}(t)$ as a shorthand for $\mathbf{mod}(\underline{t})$, for foreign keys to table t. Tabular expressions are just Fun expressions extended with a construct $E : t.c$ to dereference columns of other tables. In $E : t.c$ we expect that $E : \mathbf{link}(t)$ and c is a column of table t.

A Tabular schema is a sequential declaration of named tables. Tables are sequential declarations of named **static** or **inst** level attributes. The definition of a **input** attribute must be the empty model ε; other attributes must have a model that is a proper expression E.

Tabular Schemas:

$\mathbb{S} ::= [(t_1 = \mathbb{T}_1); \ldots; (t_n = \mathbb{T}_n)]$	(database) schema
$\mathbb{T} ::= [\mathsf{col}_1; \ldots; \mathsf{col}_n]$	table (or function)
$\mathsf{col} ::= (c : T \ \ell \ viz \ M)$	attribute c declaration
$viz ::= \mathbf{input} \mid \mathbf{local} \mid \mathbf{output}$	visibility
$M ::= \varepsilon \mid E$	model expression

7.2 Fabular: Extending Tabular with Formulas

We endow Tabular with regression syntax by extending the syntax of model expressions with regression formulas (and predictors):

Fabular model expressions:

$M ::= \ldots \mid \sim r$	model expression

[2] However, there is one subtle point of difference. In `lmer`, all coefficients in a single random effects term are assumed to be correlated, e.g., in $(1 + x + y|c)$, it is assumed that, *a priori*, the intercept 1 and slope terms for x and y have some correlation (Bates et al. 2014). This default behavior can be overridden using a double bar, e.g., $(1 + x + y||c)$. Thus, the single bar in our calculus actually corresponds to the double bar in `lmer`.

The meaning of a well-typed regression attribute is given by a simple translation to Tabular:

Translation of Regressions to Tabular: $[\![r]\!]\ \vec{e}\ \vec{f}\ \vec{F} = (\mathbb{T}, E)$

$$
\begin{aligned}
&[\![r]\!]\ \vec{e}\ \vec{f}\ \vec{F} \triangleq (\mathbb{T}, [\textbf{for}\ \vec{z} < \vec{e}\ \rightarrow\ E]) \quad \text{where } (\mathbb{T}, E) = [\![r]\!]^{\dagger}\ \vec{f}\ \vec{F} \\
&\text{and} \\
&[\![D(u_1, \ldots, u_n)]\!]^{\dagger}\ \vec{f}\ \vec{F} \triangleq ((), D([\![u_1]\!]\ \vec{z}, \ldots, [\![u_n]\!]\ \vec{z})) \\
&[\![v\{\alpha \sim r\}]\!]^{\dagger}\ \vec{f}\ \vec{F} \triangleq \text{let } (\mathbb{T}, E) = [\![r]\!]\ \vec{f}\ []\ []\ \text{in} \\
&\qquad\qquad (\mathbb{T}@[(\alpha : \textbf{real}[\vec{f}]\ \textbf{static output}\ E)], \\
&\qquad\qquad [\![v]\!]\ \vec{z} \times \alpha[\vec{F}]) \\
&[\![r + r']\!]^{\dagger}\ \vec{f}\ \vec{F} \triangleq \text{let } (\mathbb{T}, E) = [\![r]\!]^{\dagger}\ \vec{f}\ \vec{F}\ \text{in} \\
&\qquad\qquad \text{let } (\mathbb{T}', E') = [\![r']\!]^{\dagger}\ \vec{f}\ \vec{F}\ \text{in} \\
&\qquad\qquad (\mathbb{T}@\mathbb{T}', E + E') \\
&[\![r \mid v]\!]^{\dagger}\ \vec{f}\ \vec{F} \triangleq [\![r]\!]^{\dagger}(f :: \vec{f})\ (F :: \vec{F}) \\
&\qquad\qquad \text{where } \Gamma; \vec{e} \vdash v : \textbf{mod}(f) \text{ and } F = [\![v]\!]\ \vec{z} \\
&[\![(v\alpha)r]\!]^{\dagger}\ \vec{f}\ \vec{F} \triangleq \text{let } (\mathbb{T}, E) = [\![r]\!]^{\dagger}\ \vec{f}\ \vec{F}\ \text{in} \\
&\qquad\qquad \text{let } \mathbb{T}' = [\text{for } (\beta : T\ \ell\ viz\ F)\ \text{in } \mathbb{T} \rightarrow \\
&\qquad\qquad\qquad (\beta : T\ \ell\ (\text{if } \beta = \alpha \text{ then } \textbf{local} \text{ else } viz)\ F)] \\
&\qquad\qquad \text{in } (\mathbb{T}', E)
\end{aligned}
$$

Attribute c Defined by Regression Formula r:

$$
\begin{aligned}
&((c : \textbf{real inst}\ viz\ \sim r))^{\dagger} \triangleq \mathbb{T}@[(c : \textbf{real inst}\ viz\ E)]) \\
&\qquad \text{where } [\![r]\!]\ []\ []\ [] = (\mathbb{T}, E) \text{ and } viz \neq \textbf{input}
\end{aligned}
$$

The Fabular translation function $[\![r]\!]\ \vec{e}\ \vec{f}\ \vec{F}$ mirrors our earlier translation to Fun, but now returns a pair of a Tabular table $\mathbb{T}$ and an expression E. The table binds the parameters introduced by the regression to **static** attributes; the expression E is the body of the regression that is used as the model of the **inst**-level attribute c. Though syntactically different, this translation is semantically equivalent to our previous semantics—it is re-factored to introduce a single nested loop per regression rather than one nested loop per regression term. The local function $[\![r]\!]^{\dagger}$ constructs the body of this loop, for fixed loop variables $\vec{z}$ with bounds $\vec{e}$. This scheme yields more legible code but does not affect (nor improve) the computational complexity of the model. One subtlety is that setting $\vec{e}$ to the empty list in the initial call to $[\![r]\!][][][]$ ensures that local predictors are referenced directly and *not* inappropriately indexed.

For Fabular, the translation of variable predictors must also be slightly adjusted. The translation is induced by the Tabular context — it is the identity on locally declared attributes, introduces a static reference $(t.c)$ for a predictor c declared statically in table t, and an instance reference $(E : t.c)$ for a predictor c declared at instance level in table t. We elide the details.

7.3 Implementation of Fabular

We have an implementation of Fabular that includes support for vectorized regressions and syntactic sugar (not shown here) to omit parameter names and default priors on coefficients and noise. We have successfully run Fabular on a variety of models, including all of the ones mentioned in the paper. The add-in allows the user to selectively reduce Fabular regression to equivalent Tabular programs as well as extract auto-generated C# code to construct the Infer.NET model. The following table shows the code expansions (measured in LOC) from Fabular to Tabular to C# Infer.NET modeling code on a selection of models, together with the runtime for inference. We use the auto-generated code as a proxy for the length of the manually coded, equivalent Infer.NET model.

Matchbox was run on a subset of the MovieLens dataset with 6040 users, 3883 movies and the first 10000 of the available 1 million ratings. The LinearClassifier was run on a small table of 200 points. The Radon dataset comprised 87 counties and 919 houses. The Cheese model is the sales volume of sliced cheese of 5555 stores in 46 cities belonging to 50 chains. The Elections model is a classic hierarchical regression discussed by Gelman and Hill (2007) and predicts the outcome of a US election given 561 past state election outcomes across 50 states and 12 years.

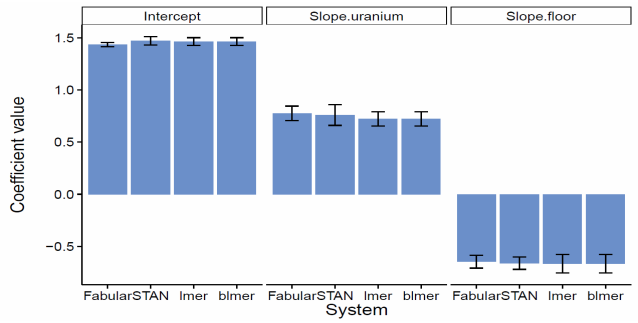


Figure 1. Radon model results. For Fabular/STAN, error bars indicate standard deviation of the posterior. For `lmer`/`blmer`, they indicate standard error.

Model	Fabular	Tabular	C#	Runtime
MatchBox	30	37	522	5s
LinearClassifier	6	9	130	1.5s
Radon	6	13	117	1.5s
Cheese	9	13	99	6.7s
Elections	31	53	373	2.5s

We compared Fabular, which performs inference with expectation propagation (EP), with three other systems: `STAN` (Stan Development Team 2014b), which implements an HMC-based method called NUTS; `lmer` (Bates et al. 2014), an R package that performs maximum likelihood estimation for hierarchical linear models; and `blmer` (Dorie 2014), which augments `lmer` with Bayesian priors.

We compared these in modelling a data set taken from Gelman and Hill (2007) that contains radiation measurements taken in houses across different counties in Minnesota. We use the hierarchical model r_7 from Section 4.1, which defines radiation activity in terms of the floor of the measurement (basement versus first floor) and the county. Not directly runnable in `lmer` or `blmer`, we reduced the model to `activity` $\sim$ `floor+(1|county)+uranium`.

The `floor` coefficients in the Fabular/STAN and classical forms are directly comparable. We compared the $1\{\alpha \sim 1 + \mathsf{u} + ?\} \mid \mathsf{c}$ term for Fabular/STAN with the `uranium` term of `lmer`/`blmer` by comparing the county-intercept-versus-uranium slope for Fabular/STAN with the activity-versus-uranium slope for `lmer`/`blmer`. Estimates of model coefficients are shown in Figure 1. Observe that Fabular produces similar results to the other three systems. This indicates that the expectation propagation algorithm of Infer.NET gives viable results for inference in hierarchical models.

7.4 Example: Generalized Linear Models

A *generalized* linear model passes its continuous output through a *link* function; for example, in logistic regression the output passes through an non-linear *logistic* function to produce a sigmoidal output. The regression calculus does not include link-functions on outputs, although it would be a simple extension.

Adding regressions to Tabular extends our reach to such generalized linear models. Take for example:

table Data

X1	**real**	**input**	
…			
X6	**real**	**input**	
Z	**real**	**output**	∼X1 + X2 + X3+ X4 + X6 + 1.0
Y	**bool**	**output**	Z > 0.0 //a **link function**

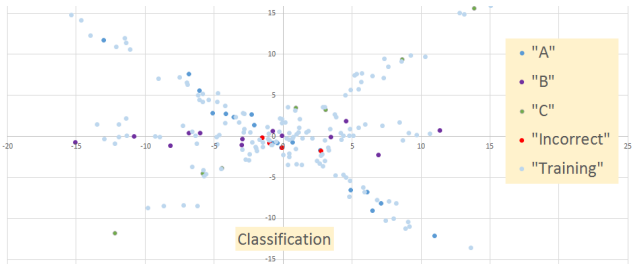
Table Data consists of six real-valued clinical measurements X1 trhough X6 and a Boolean label Y to be predicted (given some labelled training points). This model is an instance of the Bayes Point Machine (Minka 2001), a boolean classifier, in which the prior over an implicit weight vector is drawn from a vector of Gaussian priors, and the label Y is generated by thresholding a latent score Z (the inner product of the weight and input vectors). (Intuitively, Z is the distance of the point X1,..,X6 from the hyperplane defined by the weight vector; Z's sign determines the side of the hyperplane occupied by the point). Here, the weight vector is implicitly defined by the parameters of the regression; thresholding is the link function. (Gordon et al. 2015) describe a more verbose variant of this model.

7.5 Example: Latent Variable Mixture Model

Fabular goes beyond traditional regression. Predictors are no longer restricted to deterministic input data: they can be partially or even completely unobserved random variables. For example, consider the task of regressing points belonging to a mixture of various lines. If each point is labelled as belonging to a given class then we can use the label as the predictor that groups a family of linear regressions and infer the parameters of each line.

However, we can also generalize this model by assuming that only a subset of the points have been labelled. In Figure 2, we explicitly impose a discrete distribution on each point's Class. The figure contains both the Fabular code and its expansion to Core Tabular. The result is a supervised *classifier*, that infers the most likely class that each *unlabelled* point belongs to. (The operator | has lower precedence than + so the entire sum is grouped by Class.)

For example, here is a plot we generated using synthetic data from a mixture of three lines (training data points in light blue).



The blue, purple and green points are correctly classified, but the red ones were incorrectly classified. The red points all lie close to the intersections of the lines and are thus harder to separate.

If we furthermore assume that *none* of the points are labelled, then we obtain an unsupervised *clustering* algorithm, that partitions the points into three distinct sets of similar points.

7.6 Extension: Vectorized Regression

Until now our predictors have had scalar types, but our notation extends naturally to vectorized regressions, whose predictors have *vector* types and contain *arrays* of scalars. The vectorized notation is convenient shorthand for the simultaneous definition of a family of regressions. Our Matchbox example, presented in the sequel, illustrates the feature. To generalize the notation, we index our regression judgments by the dimensionality d of the regression. The scalar dimensionality $\bullet$ indicates a regression or predictor producing a single scalar value (as before). The vector dimensionality $\bullet[n]$ indicates a regression or predictor producing an n-vector of scalar values. We also define the operation of *dimensioning* a type, written d(T): $\bullet(T) \triangleq T$ is just the identity on types while $\bullet[n](T) \triangleq T[n]$, vectorizing its type argument.

Dimensionalities of Types:

$d ::= \bullet \mid \bullet[n]$	scalar or vector dimensionality

The typing rules for regressions are indexed by an additional dimensionality d and enforce that the dimensions of subregressions and coefficients are invariant and thus consistent.

Judgments of the Vectorized Type System:

$\Gamma;\vec{e} \vdash^d v : T$	d-dimensional predictor has type T
$\Gamma;\vec{e};\vec{f} \vdash^d r \,!\, \Pi$	d-dimensional regression r exports Π

Most rules merely propagate the invariant into sub-expressions. Rule (COEFF-VEC) generalizes rule (COEFF): it requires the type of the predictor expression to match the expected dimension d and vectorizes the parameters α when d is a vector. Rule (SUM-VEC) restricts its terms to have the same dimensionality.

Typing Rules for Vectorized Regressions (extract):

(COEFF-VEC)

$$\frac{\Gamma;\vec{e} \vdash^d v : d(\textbf{real}) \qquad \Gamma;\vec{f};[] \vdash^d r \,!\, \Pi \quad \alpha \notin \mathrm{dom}(\Gamma,\Pi)}{\Gamma;\vec{e};\vec{f} \vdash^d v\{\alpha \sim r\} \,!\, (\Pi, \alpha : d(\textbf{real})[\vec{f}])}$$

(SUM-VEC)

$$\frac{\Gamma;\vec{e};\vec{f} \vdash^d r \,!\, \Pi \qquad (\Gamma,\Pi);\vec{e};\vec{f} \vdash^d r' \,!\, \Pi'}{\Gamma;\vec{e};\vec{f} \vdash^d r + r' \,!\, (\Pi,\Pi')}$$

Full typing judgments are presented in (Borgström et al. 2015).

The details of the vectorizing translation are straightforward but omitted for the sake of brevity. Constant predictors are replicated as vectors of constants; vectorized coefficient terms translate to vectors of component-wise products and sums of vectorized regressions translate to component-wise sums of vectors.

In our Fabular implementation, the decision to vectorize a regression is driven by the type of the attribute defined by that regression. For added convenience, scalar predictors appearing in a context expecting a vector are implicitly cast into the appropriately sized vector (by replication).

7.7 Example: Matchbox

Matchbox (Stern et al. 2009) is a recommender system that works by integrating metadata about users and items. The system represents users and items as *trait vectors* in a common latent space and defines the *affinity* between users and items as the dot product of their vectors. Matchbox then uses this affinity to predict the ratings that a user would give to an item. We demonstrate that, to our surprise, this matrix model of recommendations can be concisely captured as a Fabular vectorized regression.

In Matchbox, $\mathbf{u} \in \mathbb{R}^n$ and $\mathbf{m} \in \mathbb{R}^m$ are sparse binary vectors representing metadata for users and items, respectively. For illustration, we might have $n = 3$ users and $m = 2$ movies as items:

$$\mathbf{u} = \begin{matrix}\mathsf{userId}_0 \\ \mathsf{userId}_1 \\ \mathsf{userId}_2\end{matrix}\begin{pmatrix}0\\0\\1\end{pmatrix} \qquad \mathbf{m} = \begin{matrix}\mathsf{movieId}_0 \\ \mathsf{movieId}_1\end{matrix}\begin{pmatrix}1\\0\end{pmatrix}$$

Thus, rows are labelled by *values* for dimensions of the metadata, e.g., $\mathbf{u}$ has one row per user id, while $\mathbf{m}$ has one row per movie id. We project $\mathbf{u}$ and $\mathbf{m}$ into a common trait space by left-multiplying them by random matrices UT and MT, which have dimensions $k \times n$ and $k \times m$, yielding $\mathbf{ut} = UT\mathbf{u}$ and $\mathbf{mt} = MT\mathbf{m}$, e.g., for $k = 2$ and trait space $\mathbb{R}^2$:

$$\mathbf{ut} = \begin{pmatrix} UT_{00} & UT_{01} & UT_{02} \\ UT_{10} & UT_{11} & UT_{12} \end{pmatrix} \times \begin{pmatrix}0\\0\\1\end{pmatrix} = \begin{pmatrix} UT_{02} \\ UT_{12} \end{pmatrix}$$

Where the individual UT_{ij} and MT_{ij} components are drawn from independent Gaussians (note that we sample a single UT and MT

table Classes
table Points

X	**real**	**input**	
Class	**link**(Classes)	**output**	DiscreteUniform(SizeOf(Classes))
Y	**real**	**output**	∼(1.0{intercept∼Gaussian(0.0,100.0)} + X{slope∼Gaussian(0.0,100.0)} + ?{pi∼Gamma(1.0,2.0)}|Class)

table Classes
table Points

X	**real**	**input**	
Class	**link**(Classes)	**output**	DiscreteUniform(SizeOf(Classes))
intercept	**real**[SizeOf(Classes)]	**static output**	[**for** c1 < SizeOf(Classes)-> Gaussian(0.0,100.0)]
slope	**real**[SizeOf(Classes)]	**static output**	[**for** c2 < SizeOf(Classes)-> Gaussian(0.0,100.0)]
pi	**real**[SizeOf(Classes)]	**static output**	[**for** c3 < SizeOf(Classes)-> Gamma(1.0,2.0)]
Y	**real**	**output**	1.0∗intercept[Class] + X∗slope[Class] + 0.0 + GaussianFromMeanAndPrecision(0.0,pi[Class])

Figure 2. Linear Classification as a regression in Fabular (and its expansion to Tabular)

table Users

gender	**link**(Genders)	**input**	
threshold1	**real**	**output**	Gaussian(-1.5,1.0)
threshold2	**real**	**output**	Gaussian(-0.5,1.0)
threshold3	**real**	**output**	Gaussian(0.5,1.0)
threshold4	**real**	**output**	Gaussian(1.5,1.0)
userTraitMean	**real**[2]	**output**	∼(1.0{umean∼Gaussian(0.0,1.0)}|gender)

table Movies

ID	**int**	**input**	
genre	**link**(Genres)	**input**	
traitMeanOther	**real**[2]	**output**	∼(1.0{mmean∼Gaussian(0.0,1.0)}|genre)
movieTraitMean	**real**[2]	**output**	**if** ID = 1 **then** [1.0;0.0] **else if** ID = 2 **then** [0.0;1.0] **else** traitMeanOther

table RatingQuery

userId	**link**(Users)	**input**	
movieId	**link**(Movies)	**input**	
UserTrait	**real**[2]	**output**	∼(1.0{userMean∼δuserTraitMean}|userId)
MovieTrait	**real**[2]	**output**	∼(1.0{movieMean∼δmovieTraitMean}|movieId)
affinity	**real**	**output**	Sum([**for** i < 2 -> UserTrait[i] ∗ MovieTrait[i]])
level	**real**	**output**	∼δaffinity + (1.0{userbias∼Gaussian(0.0,1.0)}|userId) + (1.0{moviebias∼Gaussian(0.0,1.0)}|movieId)+ ?{Prec∼δ10.0}
Rating1	**bool**	**output**	level > Gaussian(userId.threshold1,0.1)
Rating2	**bool**	**output**	level > Gaussian(userId.threshold2,0.1)
Rating3	**bool**	**output**	level > Gaussian(userId.threshold3,0.1)
Rating4	**bool**	**output**	level > Gaussian(userId.threshold4,0.1)

Figure 3. Matchbox, an illustration of vectorized regression (trivial (user) Genders and (movie) Genres tables omitted)

and use this for all users and movies, respectively). Because **u** and **m** are sparse binary vectors, **ut** and **mt** each denote a single *column* of UT and MT. Thus, we can view **ut** and **mt** as vector intercepts, selected by the encoded values, **u** and **m**, of the user and movie ids.

For a generic userId, we can express the trait vector componentwise using k scalar regressions or, more succinctly, as a single k-dimensional vectorized regression:

$$\mathbf{ut} = \begin{pmatrix} ut_0 \\ \cdots \\ ut_k \end{pmatrix} = \begin{pmatrix} (1\{UT_0 \sim \mathsf{Gaussian}(0,1)\}|\mathsf{userId}) \\ \cdots \\ (1\{UT_k \sim \mathsf{Gaussian}(0,1)\}|\mathsf{userId}) \end{pmatrix} = (1\{\mathbf{UT} \sim \mathsf{Gaussian}(0,1)\}|\mathsf{userId})$$

Trait vectors, the crucial data representation in Matchbox, are surprisingly compact in vectorized Fabular.

A more realistic model exploits features of the entities, such as the gender of users or the genre of movies. The features are used to impose pooled priors on the Gaussian trait vectors, grouped by feature. The pooled prior addresses the cold-start problem by assuming that new users behave like other users of the same gender:

$$\begin{aligned} \mathbf{utm} &= (1\{\mathbf{um} \sim \mathsf{Gaussian}(0,1)\}|\mathsf{gender}) \\ \mathbf{ut} &= (1\{\mathbf{UT} \sim \delta\mathbf{utm}\}|\mathsf{userId}) \end{aligned}$$

See Figure 3 for the full model, which exploits features in this way but also includes bias terms for individuals users and movies. The model also adds an ordinal regression response that thresholds ratings to a 5 point scale — illustrating a link function.

8. Conclusions

We set out to enrich probabilistic programming languages with the regression formulas from R. Our regression calculus amounts to an explicit description of the underlying syntax and (probabilistic) semantics of R formulas. We embed the calculus in Tabular, and hence allow users of a particular probabilistic programming system to write models with formulas. They go beyond R in setting priors for coefficients, using latent variables, tolerating missing input values, and getting the benefits of efficient inference using Infer.NET.

Tabular seeks to empower spreadsheet users with probabilistic models. By incorporating R's popular formula notation, we hope Fabular makes it easier for spreadsheet users to get started with modelling, as they need only specify a single formula. In future work, we aim to develop a graphical user interface to ease the construction of Fabular formulas, and to help configure priors.

And formulas would make a great feature for other languages!

Acknowledgments We are grateful for useful discussions with Aditya Nori and Tom Minka. Adam Ścibior received travel support from the DARPA PPAML programme. Marcin Szymczak was supported by Microsoft Research through its PhD Scholarship Programme.

References

D. Bates, M. Mächler, B. Bolker, and S. Walker. Fitting Linear Mixed-Effects Models using lme4. *ArXiv*, 2014. arXiv:1406.5823 [stat.CO].

S. Bhat, J. Borgström, A. D. Gordon, and C. V. Russo. Deriving probability density functions from probabilistic functional programs. In N. Peterman and S. Smolka, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'13)*, volume 7795 of *Lecture Notes in Computer Science*, pages 508–522. Springer, 2013.

J. Borgström, A. D. Gordon, M. Greenberg, J. Margetson, and J. V. Gael. Measure transformer semantics for Bayesian machine learning. *Logical Methods in Computer Science*, 9(3), 2013. Preliminary version at ESOP'11.

J. Borgström, A. D. Gordon, L. Ouyang, C. Russo, A. Ścibior, and M. Szymczak. Fabular: Regression formulas as probabilistic programming. Technical Report MSR–TR–2015–83, Microsoft Research, 2015.

V. Dorie. *Mixed Methods for Mixed Models*. PhD thesis, Columbia University, 2014.

A. Gelman and J. Hill. *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge University Press, 2007.

W. R. Gilks, A. Thomas, and D. J. Spiegelhalter. A language and program for complex Bayesian modelling. *The Statistician*, 43:169–178, 1994.

N. Goodman, V. K. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum. Church: a language for generative models. In *Uncertainty in Artificial Intelligence (UAI'08)*, pages 220–229. AUAI Press, 2008.

N. D. Goodman. The principles and practice of probabilistic programming. In *Principles of Programming Languages (POPL'13)*, pages 399–402, 2013.

A. D. Gordon, M. Aizatulin, J. Borgström, G. Claret, T. Graepel, A. Nori, S. Rajamani, and C. Russo. A model-learner pattern for Bayesian reasoning. In *Principles of Programming Languages (POPL'13)*, 2013.

A. D. Gordon, T. Graepel, N. Rolland, C. V. Russo, J. Borgström, and J. Guiver. Tabular: a schema-driven probabilistic programming language. In *Principles of Programming Languages (POPL'14)*, 2014a.

A. D. Gordon, T. A. Henzinger, A. V. Nori, and S. K. Rajamani. Probabilistic programming. In *Future of Software Engineering (FOSE 2014)*, pages 167–181, 2014b.

A. D. Gordon, C. V. Russo, M. Szymczak, J. Borgström, N. Rolland, T. Graepel, and D. Tarlow. Probabilistic programs as spreadsheet queries. In J. Vitek, editor, *Programming Languages and Systems (ESOP 2015)*, volume 9032 of *Lecture Notes in Computer Science*, pages 1–25. Springer, 2015.

R. Hahn. Statistical formula notation in R. URL `http://faculty.chicagobooth.edu/richard.hahn/teaching/FormulaNotation.pdf`.

O. Kiselyov and C. Shan. Embedded probabilistic programming. In *Conference on Domain-Specific Languages*, volume 5658 of *Lecture Notes in Computer Science*, pages 360–384. Springer, 2009.

D. Lunn, C. Jackson, N. Best, A. Thomas, and D. Spiegelhalter. *The BUGS Book*. CRC Press, 2013.

V. Mansinghka, D. Selsam, and Y. Perov. Venture: a higher-order probabilistic programming platform with programmable inference. *CoRR*, 2014. arXiv:1404.0099v1 [cs.AI].

B. Milch, B. Marthi, S. J. Russell, D. Sontag, D. L. Ong, and A. Kolobov. *Statistical Relational Learning*, chapter BLOG: Probabilistic Models with Unknown Objects. MIT Press, 2007.

T. Minka, J. Winn, J. Guiver, and A. Kannan. Infer.NET 2.3, Nov. 2009. Software available from `http://research.microsoft.com/infernet`.

T. P. Minka. *A family of algorithms for approximate Bayesian inference*. PhD thesis, Massachusetts Institute of Technology, 2001.

F. Morandat, B. Hill, L. Osvald, and J. Vitek. Evaluating the design of the R language - objects and functions for data analysis. In J. Noble, editor, *ECOOP 2012 - Object-Oriented Programming*, volume 7313 of *Lecture Notes in Computer Science*, pages 104–131. Springer, 2012.

A. V. Nori, C.-K. Hur, S. K. Rajamani, and S. Samuel. R2: An efficient MCMC sampler for probabilistic programs. In *Conference on Artificial Intelligence*. AAAI, July 2014.

B. Paige and F. Wood. A compilation target for probabilistic programming languages. In *ICML*, 2014.

A. Pfeffer. Figaro: An object-oriented probabilistic programming language. Technical report, Charles River Analytics, 2009.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015. URL `http://www.R-project.org/`.

S. R. Riedel, S. Singh, V. Srikumar, T. Rocktäschel, L. Visengeriyeva, and J. Noessner. WOLFE: strength reduction and approximate programming for probabilistic programming. In *Statistical Relational Artificial Intelligence (StarAI 2014)*, volume WS-14-13 of *AAAI Technical Report*. The AAAI Press, 2014.

Stan Development Team. Stan: A C++ library for probability and sampling, version 2.2, 2014a. URL `http://mc-stan.org/`.

Stan Development Team. RStan: the R interface to Stan, version 2.5.0, 2014b. URL `http://mc-stan.org/rstan.html`.

D. H. Stern, R. Herbrich, and T. Graepel. Matchbox: large scale online Bayesian recommendations. In J. Quemada, G. León, Y. S. Maarek, and W. Nejdl, editors, *Proceedings of the 18th International Conference on World Wide Web (WWW 2009)*, pages 111–120. ACM, 2009.

S. E. Whaley, M. Sigman, C. Neumann, N. Bwibo, D. Guthrie, R. E. Weiss, S. Alber, and S. P. Murphy. The impact of dietary intervention on the cognitive development of Kenyan school children. *The Journal of Nutrition*, 133(11):3965S–3971S, 2003.

F. Wood, J. W. van de Meent, and V. Mansinghka. A new approach to probabilistic programming inference. In *Proceedings of the 17th International conference on Artificial Intelligence and Statistics*, volume 33 of *JMLR Workshop and Conference Proceedings*, 2014. arXiv:1403.0504v2 [cs.AI].