“Statistical thinking will one day be as necessary for efficient citizenship as the ability to read and write.”
H.G. Wells
Empowering Spreadsheet Users with Probabilistic Programming

Andrew D. Gordon
(MSR and Edinburgh)

#OPLSS, Eugene, Oregon

Based on joint work with Johannes Borgström (Uppsala), Thore Graepel (MSR), Long Ouyang (Stanford), Nicolas Rolland (MSR), Claudio Russo (MSR), Adam Scibior (Cambridge), Marcin Szymczak (Edinburgh), and Danny Tarlow (MSR)

@AndrewDGordon
aka.ms/CalcIntel
aka.ms/popl
Probabilistic Programming

• Start with your favourite paradigm (fun, logic, imp), add random to get probabilistic behaviour, add constraints to condition on observed data, and indicate which variables’ distributions to be inferred.

• Better than writing code for probabilistic inference from scratch.

• Many academic and commercial systems: BUGS, Figaro, pcc, Church, Dimple, Hansei, STAN, Infer.NET/Fun, R2, Factorie, BLOG, ProbLog, Alchemy, Venture, Anglican, Wolfe, Hakaru, Edward, Gen, ....

• Academic prizes (eg in computer vision), and large-scale adoption in some commercial services (eg Microsoft Office Clutter, TrueSkill, etc)

• Semantics pioneered by Kozen, Giry, Jones/Plotkin, Panangaden et al, Ramsey/Pfeffer
Probabilistic Programming Languages since 1989

1989

Church: a language for generative models

Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz & Joshua B. Tenenbaum
MIT BCS/CSAIL
Cambridge, MA 02139

2004

2007

2010?
Plan

• Monday:  
  Probabilistic Programming in Infer.NET and Tabular
• Tuesday:  
  Probabilistic Programming in Fabular and Stan
• Wednesday:  
  End-User Probabilistic Programming

• Purpose:
  • Why we are doing this? Who are we doing it for?
  • An intuition of how we are doing this
  • Some challenges for the future
A simple example

(slides in this section adapted from a talk by Tom Minka)
A simple example

I secretly toss two coins

I tell you that they are NOT both heads.
What is the probability that the first coin is heads?
How do we specify the model?

• Probability distribution
  • most flexible, dense, opaque

• Factor graph
  • less flexible, clear (for small graphs)

• Simulator
  • more flexible, clearer for large graphs
Model for a fair coin

\[ p(\text{coin}) = 0.5^{\text{coin}} 0.5^{1-\text{coin}} \]  

(Bernoulli distribution)

bool coin = random() < 0.5;

(Factor graph)  

(Simulator)
Model for two coins problem

\[ p(\text{coin}_1)p(\text{coin}_2)p(\text{bothHeads}|\text{coin}_1,\text{coin}_2) \]

bool coin1 = random()<0.5;
bool coin2 = random()<0.5;
bool bothHeads = coin1 & coin2;
Bayesian inference

• Mathematical approach
  • solving integrals

• Simulation approach
  • rejection sampling

• Message-passing approach
  • local operations on factor graph
Simulation approach

```cpp
bool coin1 = random()<0.5;
bool coin2 = random()<0.5;
bool bothHeads = coin1 & coin2;
```
After many, many runs...

```c
bool coin1 = random() < 0.5; // T ~50%
bool coin2 = random() < 0.5; // F ~50%
bool bothHeads = coin1 & coin2; // T ~25%, F ~75%
```
Attaching data to the model

```c++
bool coin1 = random()<0.5;

bool coin2 = random()<0.5;

bool bothHeads = coin1 & coin2;
```

We observe that bothHeads is \( \boxed{\text{F}} \)
Throw away runs that don’t match...

```cpp
bool coin1 =
    random() < 0.5;

bool coin2 =
    random() < 0.5;

bool bothHeads =
    coin1 & coin2;
```

- `coin1` has a probability of `~33%`
- `coin2` has a probability of `~67%`
- `bothHeads` has a probability of `~0%`
Bayesian inference

• Mathematical approach
  • solving integrals

• Simulation approach
  • rejection sampling

• Message-passing approach
  • local operations on factor graph
Message passing

\[(T,F)\]

Belief propagation

(Pearl, 1982)
Efficiency

- Want to reproduce known efficient algorithms for machine learning
  - Message passing in factor graphs
- Motivates using a compiler
- Goal of compiler is to output same code that person would write for solving that model
Infer.NET
an inference engine
How Infer.NET works

Model

Transformations

Factor graph

Inference compiler

Inference steps

Inference execution

Results

Data
Coins model in Infer.NET

```csharp
var coin1 = Variable.Bernoulli(0.5); // (random()<0.5)

var coin2 = Variable.Bernoulli(0.5);

var bothHeads = coin1 & coin2;
```
Running inference in the model

```plaintext
var engine = new InferenceEngine();

Bernoulli result =
    engine.Infer<Bernoulli>(bothHeads);

// ‘result’ is Bernoulli(0.25)
```
Attaching data to the model

We observe that bothHeads is F

bothHeads.ObservedValue = false;

Bernoulli dist1 =
    engine.Infer<Bernoulli>(coin1);

// ‘dist1’ is Bernoulli(0.333...)
Empowering Spreadsheet Users

• Modern spreadsheets incorporate end-user database functionality previously in separate applications

• “A data enthusiast is an educated person who believes that data can be used to answer a question or solve a problem. These people are not mathematicians or programmers, and only know a bit of statistics.” (Pat Hanrahan 2012)

• Goal: empower data enthusiasts, not professional programmers or data scientists
Tabular

- DSL to get insight from data with probabilistic programs

- Tabular has three guiding principles:
  1) Language design based on annotations of the data schema
  2) Inference by querying latent variables and missing values
  3) Auto-suggest models based on the structure of the data

Papers at POPL’14, ESOP’15, POPL’16
Structure of a Model

A Bayesian model

\[ y = Ax + B + e \]

where noise \( e \sim N(0,2^2) \) and parameters \( A, B \sim N(0,1) \)

<table>
<thead>
<tr>
<th>A</th>
<th>real</th>
<th>static latent</th>
<th>Gaussian(0,1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>real</td>
<td>static latent</td>
<td>Gaussian(0,1)</td>
</tr>
<tr>
<td>x</td>
<td>real</td>
<td>input</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>real</td>
<td>output</td>
<td>Gaussian(A*x+B,4)</td>
</tr>
</tbody>
</table>

Parameter distribution \( p(A, B) \)

Sampling distribution \( p(\tilde{y} | A, B, \tilde{x}) \) (aka the likelihood)

Predictive distribution \( p(\tilde{y}|\tilde{x}) \triangleq \int p(A, B) \ p(\tilde{y} | A, B, \tilde{x}) \ d(A, B) \)

With no observed outputs, these are the prior distributions
Conditioned on (some) outputs, we obtain the posterior distributions
Probabilistic Inference

- Monte Carlo
  - Set of samples for joint
  - Slow but accurate

- Message Passing
  - Parameters for marginals
  - Fast but approximate

Stanisław Ulam

Judea Pearl
Factor Graphs

\[ p(A, B, \hat{x}, \hat{y}) = pdf(A, Gaussian(0,1)) \times pdf(B, Gaussian(0,1)) \times \prod_i pdf(y_i, Gaussian(Ax_i + B, 4)) \]
The Tabular Recipe

1. Start with the schema
2. Add latent columns
3. Write models for latent and output columns
4. Learn latent columns and table parameters, and/or
5. Predict missing values in output columns

• We focus on this part of the cycle of learning from data, leaving gathering, preprocessing, visualizing to other tools
TrueSkill – Data

<table>
<thead>
<tr>
<th>Players</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>String</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Matches</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Player1</td>
<td>Link(Players)</td>
</tr>
<tr>
<td>Player2</td>
<td>Link(Players)</td>
</tr>
<tr>
<td>Win1</td>
<td>Bool</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Alice</td>
</tr>
<tr>
<td>1</td>
<td>Bob</td>
</tr>
<tr>
<td>2</td>
<td>Cynthia</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Player1</th>
<th>Player2</th>
<th>Win1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>False</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>False</td>
</tr>
</tbody>
</table>
**TrueSkill**

- **Model**
- **Players**
- **Name**
- **Skill**
  - **real**
  - **latent**
  - **Gaussian(25.0,100.0)**

**Matches**

- **Player1**
  - **link(Players)**
  - **input**
- **Player2**
  - **link(Players)**
  - **input**

**Perf1**
- **real**
- **latent**
- **Gaussian(Player1.Skill, 100.0)**

**Perf2**
- **real**
- **latent**
- **Gaussian(Player2.Skill, 100.0)**

**Win1**
- **bool**
- **output**
- **Perf1 > Perf2**

**Diagram**

- **ID<SizeOf(Players)**
  - **(Gaussian(25.0,100.0))**

- **ID<SizeOf(Matches)**
  - **real Skill**
  - **upto(SizeOf(Players)) Player2**
  - **upto(SizeOf(Players)) Player1**
  - **(Gaussian((Player2).Skill,100.0))**
  - **(Gaussian((Player1).Skill,100.0))**
  - **real Perf2**
  - **real Perf1**
  - **((Perf1) > (Perf2))**
  - **bool Win1**
Query-by-Latent Column

<table>
<thead>
<tr>
<th>Name</th>
<th>Player1</th>
<th>Player2</th>
<th>Win1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Skill</th>
<th>Perf1</th>
<th>Perf2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Gaussian(20.25, 82.3)</td>
<td>Gaussian(29.75, 123.6)</td>
</tr>
<tr>
<td>1</td>
<td>Gaussian(25.0, 70.7)</td>
<td>Gaussian(34.51, 129.1)</td>
</tr>
<tr>
<td>2</td>
<td>Gaussian(29.75, 82.3)</td>
<td>Gaussian(15.49, 129.1)</td>
</tr>
</tbody>
</table>
Query-by-Missing-Value

<table>
<thead>
<tr>
<th>Name</th>
<th>Player1</th>
<th>Player2</th>
<th>Win1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Cynthia</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Bernoulli(0.31)
### Leagues

<table>
<thead>
<tr>
<th>Leagues</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>leaguesId</td>
<td>int</td>
<td>input</td>
<td></td>
</tr>
<tr>
<td>skillModifier</td>
<td>real</td>
<td>latent Gaussian(0.0, 1.0)</td>
<td></td>
</tr>
</tbody>
</table>

### Teams

<table>
<thead>
<tr>
<th>Teams</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>team1Id</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>leagueId</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>individualSkill</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>skill</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Matches

<table>
<thead>
<tr>
<th>Matches</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>homeSkillAdvantage</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>game1Id</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>team1Id</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>team1Score</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>team2Id</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>team2Score</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>team1WasHome</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>team1Home Advantage</td>
<td>real</td>
<td>latent</td>
<td>team1WasHome * homeSkillAdvantage</td>
</tr>
<tr>
<td>team1Perf</td>
<td>real</td>
<td>latent</td>
<td>Gaussian(team1ld.skill + team1HomeAdvantage, 0.25)</td>
</tr>
<tr>
<td>team2Perf</td>
<td>real</td>
<td>latent</td>
<td>Gaussian(team2ld.skill, 0.25)</td>
</tr>
<tr>
<td>team1Won</td>
<td>bool</td>
<td>output</td>
<td>team1Perf &gt; team2Perf</td>
</tr>
</tbody>
</table>
Ex: Cold Boot Attacks

• DRAMs hold their values for long intervals without power or refresh.

Hence, sensitive information such as cryptographic keys can be extracted from memory, despite OS attempts to protect its contents.
Recovering DES Keys

- The DES key schedule algorithm produces 16 subkeys, each a permutation of a 48-bit subset of bits from the original 56-bit key.
- Every bit of original key repeated in about 14 of 16 subkeys.
- Model the decay as a binary asymmetric channel
  - the probability of a 1 remaining 1 is some fixed Decay_{11}
  - the probability of a 0 flipping to a 1 is some fixed Decay_{01}.
- A probabilistic model can learn these parameters as well as estimating the latent key bits.
Bernoulli and Beta Distributions

- **Bernoulli**($p$) is the discrete distribution that returns true with probability $p$, returns false with probability $1 - p$

- **Beta**($\alpha, \beta$) is the continuous distribution over the bias $p$ of a *Bernoulli*($p$), after observing true $\alpha - 1$ times, and false $\beta - 1$ times.

- For example, *Beta*(1,1) is the uniform distribution.

Tabular Demo

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>Param</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decay01</td>
<td>real</td>
<td>param</td>
<td>Beta(1.0,1.0)</td>
</tr>
<tr>
<td>Decay11</td>
<td>real</td>
<td>param</td>
<td>Beta(1.0,1.0)</td>
</tr>
<tr>
<td>Bit</td>
<td>bool</td>
<td>output</td>
<td>Bernoulli(0.5)</td>
</tr>
<tr>
<td>Copy0</td>
<td>bool</td>
<td>output</td>
<td>if Bit then Bernoulli(Decay11) else Bernoulli(Decay01)</td>
</tr>
<tr>
<td>Copy1</td>
<td>bool</td>
<td>output</td>
<td>if Bit then Bernoulli(Decay11) else Bernoulli(Decay01)</td>
</tr>
<tr>
<td>Copy2</td>
<td>bool</td>
<td>output</td>
<td>if Bit then Bernoulli(Decay11) else Bernoulli(Decay01)</td>
</tr>
<tr>
<td>Copy3</td>
<td>bool</td>
<td>output</td>
<td>if Bit then Bernoulli(Decay11) else Bernoulli(Decay01)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decay01</td>
<td>Beta(1.524,11.75)[mean=0.1148]</td>
</tr>
<tr>
<td>Decay11</td>
<td>Beta(21.04,31.5)[mean=0.4004]</td>
</tr>
</tbody>
</table>
DARE (Bachrach et. al 2012)

- Data drawn from multiple choice intelligence test
- 60 questions with 8 possible answers (one correct)
- 121 participants
- Responses relate ParticipantID and QuestionID to Participant’s answers.
Ability

Log evidence -7499.92

Probit(a,b) = Gaussian(a,b)>0.0
Difficulty/Ability

- Participants
  - Ability: real, latent, Gaussian(0.0,1.0)
- Questions
  - Answer: upto(8), output
- Difficulty
  - real, latent, Gaussian(0.0,1.0)
- Responses
  - ParticipantID: link(Participants)
  - QuestionID: link(Questions)
- Advantage
  - real, latent, ParticipantID.Ability - QuestionID.Difficulty
- Know
  - bool, latent, Probit(Advantage,1.0)
- Guess
  - upto(8), latent, DiscreteUniform(8)
- Answer
  - upto(8), output
  - if Know then QuestionID.Answer else Guess

Log evidence: -5932.34
Case study: Psychometrics

<table>
<thead>
<tr>
<th>Model</th>
<th>Lang.</th>
<th>LOC Model</th>
<th>LOC Inference</th>
<th>LOC Total</th>
<th>Compile (s)</th>
<th>Infer (s)</th>
<th>Model log evidence</th>
<th>Avg. log prob. test Responses</th>
<th>Avg. log prob. test Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Tabular II</td>
<td>0</td>
<td>17</td>
<td>0</td>
<td>17</td>
<td>0.39</td>
<td>0.40</td>
<td>-7499.74</td>
<td>-1.432</td>
</tr>
<tr>
<td>A</td>
<td>Infer.NET</td>
<td>73</td>
<td>45</td>
<td>20</td>
<td>138</td>
<td>0.32</td>
<td>0.38</td>
<td>-7499.74</td>
<td>-1.432</td>
</tr>
<tr>
<td>DA</td>
<td>Tabular II</td>
<td>0</td>
<td>18</td>
<td>0</td>
<td>18</td>
<td>0.39</td>
<td>0.46</td>
<td>-5933.52</td>
<td>-1.118</td>
</tr>
<tr>
<td>DA</td>
<td>Infer.NET</td>
<td>73</td>
<td>47</td>
<td>21</td>
<td>141</td>
<td>0.34</td>
<td>0.43</td>
<td>-5933.25</td>
<td>-1.118</td>
</tr>
<tr>
<td>DARE</td>
<td>Tabular II</td>
<td>0</td>
<td>19</td>
<td>0</td>
<td>19</td>
<td>0.40</td>
<td>2.86</td>
<td>-5820.40</td>
<td>-1.119</td>
</tr>
<tr>
<td>DARE</td>
<td>Infer.NET</td>
<td>73</td>
<td>49</td>
<td>22</td>
<td>144</td>
<td>0.37</td>
<td>2.8</td>
<td>-5820.40</td>
<td>-1.119</td>
</tr>
</tbody>
</table>

- Replicated Infer.NET C# by Bachrach et al 2012
- Same statistical results, much less code, slight loss in perf
## Core Syntax of Schemas

<table>
<thead>
<tr>
<th>Component</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema</td>
<td>( S ::= (t_1 \ T_1) \ldots (t_n \ T_n) )</td>
</tr>
<tr>
<td>table model</td>
<td>( T ::= (c_1 \ ty_1 \ \ell_1 \ A_1) \ldots (c_n \ ty_n \ \ell_n \ A_n) )</td>
</tr>
<tr>
<td>column type</td>
<td>( ty ::= \text{bool} \mid \text{int} \mid \text{mod}(n) \mid \text{real} \mid T[n] )</td>
</tr>
<tr>
<td>level</td>
<td>( \ell ::= \text{static} \mid \epsilon )</td>
</tr>
<tr>
<td>annotation</td>
<td>( A ::= \text{input} \mid \text{latent}(E) \mid \text{output}(E) )</td>
</tr>
<tr>
<td>expression</td>
<td>( E ::= c \mid E.c \mid f(E_1,\ldots,E_n) \mid D(E_1,\ldots,E_n) \mid \ldots )</td>
</tr>
</tbody>
</table>

- Attribute marked **static** is single parameter, else whole column
- **input** attribute conditions the model, but cannot be predicted
- **latent\((E)\)** attribute defines hidden variable by \(E\)
- **output\((E)\)** attribute conditions the model, predicted by \(E\)
### Semantics of Schemas

#### Players

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Input</th>
<th>Latent</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skill</td>
<td>real</td>
<td>latent</td>
<td></td>
<td>Gaussian(25,100)</td>
</tr>
</tbody>
</table>

#### Matches

<table>
<thead>
<tr>
<th>Player1</th>
<th>Type</th>
<th>Input</th>
<th>Latent</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perf1</td>
<td>real</td>
<td>latent</td>
<td></td>
<td>Gaussian(Player1.Skill,100)</td>
</tr>
<tr>
<td>Perf2</td>
<td>real</td>
<td>latent</td>
<td></td>
<td>Gaussian(Player2.Skill,100)</td>
</tr>
</tbody>
</table>

For each row r of Players,

\[
\text{Skill} \sim \text{Gaussian}(25,100)
\]

For each row r of Matches,

\[
\begin{align*}
\text{Player1} & := r.\text{Player1} \\
\text{Player2} & := r.\text{Player2} \\
\text{Perf1} & \sim \text{Gaussian(Players[Player1].Skill,100)} \\
\text{Perf2} & \sim \text{Gaussian(Players[Player2].Skill,100)} \\
\text{Win1} & := \text{if } (\text{Perf1} > \text{Perf2}) \\
& \text{observe } (\text{Win1} == r.\text{Win1})
\end{align*}
\]
Infer and Decision Making

Players
| Skill | real | latent | Gaussian(25.0,100.0) |

Matches
| Player1 | link(Players) | input |
| Player2 | link(Players) | input |
| Win1    | bool          | output Gaussian(Player1.Skill,100.0) > Gaussian(Player2.Skill,100.0) |

Bets
| Match | link(Matches) | input |
| Odds1 | real          | input |
| p     | real          | latent | infer.Bernoulli[].Bias(Match.Win1) |
| EU    | real          | latent | p*Odds1 - (1.0-p) |
| PlaceBet | bool | latent | EU > 0.0 |

| Matches | | | |
| Player1 | Player2 | Win1 |
| 0       | Alice   | Bob  | FALSE |
| 1       | Bob     | Cynthia | FALSE |
| 2       | Alice   | Cynthia |

| Bets | | | |
| Match | Odds1 |
| 0     | 2 4   |
| 1     | 2 2   |

| Bets | | | |
| Bet | Match | Odds1 | p | EU | PlaceBet |
| 0   | 2     | 4    | 0.309234026 0.546170128 | TRUE |
| 1   | 2     | 2    | 0.309234026 -0.072297923 | FALSE |
# Beyond the Core Syntax

<table>
<thead>
<tr>
<th>Component</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
<td>$S ::= (t_1 T_1) \ldots (t_n T_n)$</td>
</tr>
<tr>
<td>Table Model</td>
<td>$T ::= (c_1 ty_1 \ell_1 A_1) \ldots (c_n ty_n \ell_n A_n)$</td>
</tr>
<tr>
<td>Column Type</td>
<td>$ty ::= \text{bool} \mid \text{int} \mid \text{mod}(n) \mid \text{real} \mid T[n]$</td>
</tr>
<tr>
<td>Level</td>
<td>$\ell ::= \text{static} \mid \epsilon$</td>
</tr>
<tr>
<td>Annotation</td>
<td>$A ::= \text{input} \mid \text{latent}(M) \mid \text{output}(M)$</td>
</tr>
<tr>
<td>Model Expression</td>
<td>$M ::= E \mid t(h_1=E_1,\ldots,h_n=E_n) \mid M[E_{\text{index}}&lt;E_{\text{size}}]$</td>
</tr>
<tr>
<td>Expression</td>
<td>$E ::= c \mid E.c \mid f(E_1,\ldots,E_n) \mid D(E_1,\ldots,E_n) \mid \ldots$</td>
</tr>
</tbody>
</table>

- An extended syntax (that compiles to the core):
  - Functions defined by table models $T$
  - Application $t(h_1=E_1,\ldots,h_n=E_n)$ defined by substitution semantics
  - Indexed models $M[E_{\text{index}}<E_{\text{size}}]$ capture structure of many models
## Functions and Indexing, by example

![Plot of Waiting Time vs Eruption Duration](image)

### faithful

<table>
<thead>
<tr>
<th>Feature</th>
<th>Type</th>
<th>Mode</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster</td>
<td>mod(2)</td>
<td>latent</td>
<td>Discrete(2,[0.5, 0.5])</td>
</tr>
<tr>
<td>eruption_duration</td>
<td>real</td>
<td>output</td>
<td>CG()[cluster]</td>
</tr>
<tr>
<td>waiting_time</td>
<td>real</td>
<td>output</td>
<td>CG(MeanOfMean=60.0)[cluster]</td>
</tr>
<tr>
<td>assignment</td>
<td>mod(2)</td>
<td>latent</td>
<td>infer.Discrete[2].Mode(cluster)</td>
</tr>
</tbody>
</table>

![Plot of Colored Points by Cluster](image)
### Functional Programming

**function:** CG

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MeanOfMean</td>
<td>real</td>
<td>static input</td>
<td>0.0</td>
</tr>
<tr>
<td>PrecOfMean</td>
<td>real</td>
<td>static input</td>
<td>1.0</td>
</tr>
<tr>
<td>Mean</td>
<td>real</td>
<td>param</td>
<td>Gaussian(MeanOfMean, 1/PrecOfMean)</td>
</tr>
<tr>
<td>Prec</td>
<td>real</td>
<td>param</td>
<td>Gamma(1.0, 1.0)</td>
</tr>
<tr>
<td>CG</td>
<td>real</td>
<td>output</td>
<td>Gaussian(Mean, 1/Prec)</td>
</tr>
</tbody>
</table>

**faithful**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>eruption_duration</td>
<td>real</td>
<td>output</td>
<td>CG()</td>
</tr>
</tbody>
</table>

**faithful**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>real</td>
<td>param</td>
<td>Gaussian(0.0, 1.0)</td>
</tr>
<tr>
<td>Prec</td>
<td>real</td>
<td>param</td>
<td>Gamma(1.0, 1.0)</td>
</tr>
<tr>
<td>eruption_duration</td>
<td>real</td>
<td>output</td>
<td>Gaussian(Mean, 1/Prec)</td>
</tr>
</tbody>
</table>
Indexed Models

• An **indexed model** \( M[E_{\text{index}} < E_{\text{size}}] \) is a model with an array of \( E_{\text{size}} \) copies of the parameters of \( M \); the parameter to produce each output is selected by \( E_{\text{index}} \).

<table>
<thead>
<tr>
<th>faithful</th>
<th>cluster</th>
<th>mod(2)</th>
<th>latent</th>
<th>Discrete(2,[0.5, 0.5])</th>
</tr>
</thead>
<tbody>
<tr>
<td>eruption_duration</td>
<td>real</td>
<td>output</td>
<td>CG()[cluster &lt; 2]</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>faithful</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster</td>
</tr>
<tr>
<td>Mean</td>
</tr>
<tr>
<td>Prec</td>
</tr>
<tr>
<td>eruption_duration</td>
</tr>
</tbody>
</table>
Automatic Model Suggestion

Model expressions allow us to recast early work on model suggestion as probabilistic metaprogramming.
Key Features of Tabular

- Schema-driven probabilistic programming
- Unique table-based syntax for embedding in spreadsheets
- Query by missing value and latent column
- Core interpreted as factor graph structured as a Bayesian model: parameters and outputs
- Functions and indexed models capture common structure
- Post-processing for Bayesian decision-making
- Sized-types for arrays and distributions
- Type-based information-flow analysis usefully distinguishes the stochastic and deterministic parts of a probabilistic program
- R-style regression calculus for hierarchical regressions
- Papers with metatheory: POPL’14, ESOP’15, POPL’16
Language-Based Statistical Thinking

- "Statistical thinking will one day be as necessary for efficient citizenship as the ability to read and write." H.G. Wells
- Tabular seeks to empower today’s spreadsheet users to be efficient citizens, by casting statistical thinking as a PL question
- Probabilistic programming as a field empowers developers to more easily build custom machine-learing solutions
- Many research challenges:
  - Better model suggestion – automatically discover suitable statistical model for dataset
  - Better model criticism – how to convey uncertainty?
  - Better inference – exciting recent progress using Deep Neural Nets to speed up Monte Carlo inference for probabilistic programs

https://github.com/TabularLang/CoreTabular
Fabular: Regression Formulas as Probabilistic Programming

Johannes Borgström, Andrew D. Gordon, Long Ouyang, Claudio Russo, Adam Ścibior, Marcin Szymczak
Uppsala, MSR, Edinburgh, Stanford, Cambridge,
Probabilistic programming:

\[
\begin{align*}
\text{let } \alpha &= \text{Gaussian}(0, \sigma^2_{\text{large}}) \text{ in} \\
\text{let } \beta &= \text{Gaussian}(0, \sigma^2_{\text{large}}) \text{ in} \\
\text{let } \pi &= \text{Gamma}(1, \lambda_{\text{small}}) \text{ in} \\
((\alpha, \beta), [\text{for } z < \text{students} \rightarrow \alpha + x[z] \times \beta + \text{Gaussian}(0, 1/\pi)])
\end{align*}
\]

In another galaxy, R’s \textit{lm} package:

\[
\begin{align*}
y \sim 1 + x \\
y \sim x
\end{align*}
\]

Our goal: embrace and extend R’s formulas to get more succinct probabilistic programs.
A regression calculus $y \sim r$

\[
\begin{align*}
  r &::= \\
  D(v_1, \ldots, v_n) &\quad \text{noise with distribution } D \\
  v\{\alpha \sim r\} &\quad \text{predictor with coefficient}\ (\text{new wrt } R) \\
  r + r' &\quad \text{sum} \\
  r \mid v &\quad \text{grouping} \\
  (v\alpha)_r &\quad \text{restriction (scope of } \alpha \text{ is } r)\ (\text{new wrt } R)
\end{align*}
\]

\[
\begin{align*}
  u, v &::= \\
  s &\quad \text{scalar (common cases are 1 and 0)} \\
  x &\quad \text{variable (categorical or continuous)} \\
  u : v &\quad \text{interaction (multiplication)}
\end{align*}
\]
Goal: model the y column as a regression $y \sim r$ on the other columns.
(1) Pure Intercept

\[ y \sim 1\{\alpha\} \sim \text{Gaussian}(0, s_{\text{large}}^2) \]

let \( \alpha = \text{Gaussian}(0, s_{\text{large}}^2) \) in

\((\alpha, [\text{for } z < \text{students} \rightarrow 1 \times \alpha])\)

\[ \nu\{\alpha\} \triangleq \nu\{\alpha \sim \text{Gaussian}(0, s_{\text{large}}^2)\} \]

\[ \nu \triangleq (\nu\alpha)\nu\{\alpha\} \quad \text{for } \alpha \notin \text{fv}(\nu) \]
(2) Pure Noise

\[ y \sim ? \]

\[
\text{let } \pi = \text{Gamma}(1, \lambda_{\text{small}}) \text{ in } \\
(()), [\text{for } z < \text{students} \rightarrow \text{Gaussian}(0, 1/\pi)]
\]
(3) Intercept with Noise

\[ y \sim 1\{\alpha\} + ? \]

let \( \alpha = \text{Gaussian}(0, s_{\text{large}}^2) \) in

let \( \pi = \text{Gamma}(1, \lambda_{\text{small}}) \) in

\((\alpha), \left[ \text{for } z < \text{students} \rightarrow \alpha + \text{Gaussian}(0, 1/\pi) \right] \)
(4) Slope and Intercept

\[ y \sim 1 \{\alpha\} + x \{\beta\} + ? \]

let \( \alpha = \text{Gaussian}(0, s_{\text{large}}^2) \) in
let \( \beta = \text{Gaussian}(0, s_{\text{large}}^2) \) in
let \( \pi = \text{Gamma}(1, \lambda_{\text{small}}) \) in

\([(\alpha, \beta), [\text{for } z < \text{students} \rightarrow \alpha + x[z] \times \beta + \text{Gaussian}(0, 1/\pi)]]\)

\[ y \sim 1 + x \]
(5) Intercept per School

\[ y \sim (1 \{ \alpha \} \mid s) \]

let \( \alpha = [\text{for } z < \text{schools} \rightarrow \text{Gaussian}(0, s_{\text{large}}^2)] \) in

let \( \pi = \text{Gamma}(1, \lambda_{\text{small}}) \) in

\( ((\alpha), [\text{for } z < \text{students} \rightarrow \alpha[s[z]] + \text{Gaussian}(0, 1/\pi)]) \)
Intercept for each school is a function of average income?
A hierarchical model is “a regression in which the parameters – the regression coefficients – are given a probability model.” (Gelman and Hill 2007)
Summary so Far

- No prior formalisations of R-style formulas
- Unlike R formulas, this regression calculus can specify priors and also hierarchical models

<table>
<thead>
<tr>
<th>Model</th>
<th>Regression</th>
<th>Evidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure Intercept</td>
<td>1</td>
<td>fails</td>
</tr>
<tr>
<td>Pure Noise</td>
<td>?</td>
<td>-5577</td>
</tr>
<tr>
<td>Intercept With Noise</td>
<td>1</td>
<td>-3320</td>
</tr>
<tr>
<td>Slope and Intercept</td>
<td>1 + x</td>
<td>-2695</td>
</tr>
<tr>
<td>Intercept Per School</td>
<td>(1{alpha}</td>
<td>s)</td>
</tr>
<tr>
<td>Hierarchical</td>
<td>(1{alpha ~ 1 + u}</td>
<td>s) + x</td>
</tr>
</tbody>
</table>
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 \( j \).

\[
\Gamma; \vec{e} \vdash v : T \quad \text{predictor } v \text{ yields an } \vec{e}\text{-cube of } T
\]

Typing Rules for Predictors:

\[
\begin{align*}
\text{(SCALAR)} & \quad \Gamma \vdash \varnothing \quad s \in \mathbb{R} \\
\text{(VAR)} & \quad \Gamma \vdash x : T[\vec{e}] \\
\text{(INTERACT)} & \quad \Gamma; \vec{e} \vdash u : \text{real} \quad \Gamma; \vec{e} \vdash v : \text{real} \\
\Gamma; \vec{e} \vdash s : \text{real} & \quad \Gamma; \vec{e} \vdash x : T \\
\Gamma; \vec{e} \vdash (u : v) : \text{real} &
\end{align*}
\]
Translation of Predictors to Fun: $\llbracket v \rrbracket \tilde{E} = E$

$\llbracket s \rrbracket \tilde{E} \triangleq s$

$\llbracket x \rrbracket \tilde{E} \triangleq x[\tilde{E}]$

$\llbracket u : v \rrbracket \tilde{E} \triangleq [u] \tilde{E} \times [v] \tilde{E}$

Lemma 2. If $\Gamma; (e_i)_{i \in I} \vdash v : T$ and $\Gamma \vdash E_i : \text{mod}(e_i)$ for all $i \in I$ then $\Gamma \vdash \llbracket v \rrbracket (E_i)_{i \in I} : T$. 
Γ; ̇e; ̇f ⊢ r ! Π  regression r yields ̇e-cube with parameter ̇f-cubes

Typing Rules for Regressions:

(NOISE)
D : (x_1 : U_1, \ldots, x_n : U_n) → real  \quad Γ; ̇e ⊢ u_j : U_j \quad \forall j ∈ 1..n
Γ; ̇e; ̇f ⊢ D(u_1, \ldots, u_n) ! ∅

(SUM)
Γ; ̇e; ̇f ⊢ r ! Π
Γ, Π; ̇e; ̇f ⊢ r' ! Π'  \quad (Γ, Π); ̇e; ̇f ⊢ r + r' ! (Π, Π')
Γ; ̇e; ̇f ⊢ r + r' ! (Π, Π')

(GROUP)
Γ; ̇e ⊢ v : mod(f)
Γ; ̇e; (f :: ̇f) ⊢ r ! Π
Γ; ̇e; ̇f ⊢ r | v ! Π

(COEFF)
Γ; ̇e ⊢ v : real  \quad Γ; ̇f; [] ⊢ r ! Π  \quad α \notin \text{dom}(Γ, Π)
Γ; ̇e; ̇f ⊢ v\{α ∼ r\} ! (Π, α : real[̇f])
Translation of Regressions to Fun: \[[r] \bar{e} \bar{f} \bar{F} = E\]

\[
[D(u_1, \ldots, u_n)] \bar{e} \bar{f} \bar{F} \equiv ((), \text{for } \bar{z} < \bar{e} \rightarrow D([u_1] \bar{z}, \ldots, [u_n] \bar{z}))
\]

\[
[v\{\alpha \sim r\}] \bar{e} \bar{f} \bar{F} \equiv \text{let } (\text{dom}(r), \alpha) = [r] \bar{f} \text{ in }
\]

\[
(\text{dom}(r) @ [\alpha], \text{for } \bar{z} < \bar{e} \rightarrow [v] \bar{z} \times \alpha[\bar{F}])
\]

\[
[r + r'] \bar{e} \bar{f} \bar{F} \equiv \text{let } (\text{dom}(r), y) = [r] \bar{e} \bar{f} \bar{F} \text{ in }
\]

\[
\text{let } (\text{dom}(r'), y') = [r'] \bar{e} \bar{f} \bar{F} \text{ in }
\]

\[
(\text{dom}(r) @ \text{dom}(r'), \text{for } \bar{z} < \bar{e} \rightarrow y[\bar{z}] + y'[\bar{z}])
\]

\[
[r \mid v] \bar{e} \bar{f} \bar{F} \equiv [r] \bar{e} (f :: \bar{f}) (F :: \bar{F}) \quad \text{where}
\]

\[
\Gamma; \bar{e} \vdash v : \text{mod}(f) \text{ and } F = [\text{for } \bar{z} < \bar{e} \rightarrow [v] \bar{z}]
\]

\[
[(\nu \alpha) r] \bar{e} \bar{f} \bar{F} \equiv \text{let } (\text{dom}(r), y) = [r] \bar{e} \bar{f} \bar{F} \text{ in}(\text{dom}(r) \setminus \alpha, y)
\]

Theorem 1 (Type Preservation). If \( \Gamma; \bar{e}; (f_j)_{j \in J} \vdash r ! \Pi \) and
\( \Gamma \vdash F_j : \text{mod}(f_j)[\bar{e}] \) for all \( j \in J \), and \( E = [r] \bar{e} (f_j)_{j \in J} (F_j)_{j \in J} \), then we have \( \Gamma \vdash E : \text{tuple}(\Pi) \times \text{real}[\bar{e}] \).
Fabular = Tabular + Formulas

<table>
<thead>
<tr>
<th>Points</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>real</td>
<td>input</td>
</tr>
<tr>
<td>Y</td>
<td>real</td>
<td>output (\sim X + 1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Points</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>real</td>
<td>input</td>
<td></td>
</tr>
<tr>
<td>Y_0</td>
<td>real</td>
<td>param</td>
<td>Gaussian((0.0,1\times10^5))</td>
</tr>
<tr>
<td>Y_1</td>
<td>real</td>
<td>param</td>
<td>Gaussian((0.0,1\times10^5))</td>
</tr>
<tr>
<td>v__1</td>
<td>real</td>
<td>param</td>
<td>Gamma((1.0,1000000.0))</td>
</tr>
<tr>
<td>Y</td>
<td>real</td>
<td>output</td>
<td>Gaussian(((X \times Y_0) + Y_1,v__1))</td>
</tr>
</tbody>
</table>
Fabular = Tabular + Formulas

A link function maps from a continuous regression to a discrete output.
model {
  real p;
  q ~ 1 + female + black + female:black
       + age + edu + age:edu +
       (1{ (1 | region) + v_prev } | state) ;
  p <- fmax(0, fmin(1, q));
  y ~ bernoulli(p);
}
What else is in the paper?

• Nested models may be eliminated – every regression normalizes to the form: 

\[(\vec{v}\alpha)(P + N)\]

**Classes of Terms: \(N, P\)**

\[
\begin{align*}
N &:= \sum_{i=1}^{n} D_i(u_{i1}, \ldots, u_{i|D_i|}) & \text{noise} \\
P &:= \sum_{i=1}^{n} v_i\{\alpha_i \sim N_i\} \mid \vec{w}_i & \text{single-level regression}
\end{align*}
\]

• We explain the essence of the popular formula notation in R's `lm` and `lmer` packages by converting formulas to the regression calculus.
Summary of Fabular

• The **regression calculus** is the first formal calculus of regressions, with a unique recursive syntax for hierarchical models (written \( v\{\alpha \sim r\} \)), going beyond the formulas of R.

• Regressions are boilerplate components of many models.

• **Fabular** mixes potentially vectorised regressions with latent variables and link functions, scrapping some boilerplate.

• We envisage that boilerplate in other languages, such as Stan, can also be eliminated.

• Synthesis of probabilistic programs in Fabular?
Mathematical Statistics
4
Probability Theory
The Logic of Science
E. T. Jaynes

Statistical Physics
3 Machine Learning
Business and Finance
Bayesian Statistics

Statistical Rethinking
A Bayesian Course with Examples in R and Stan

Richard McElreath