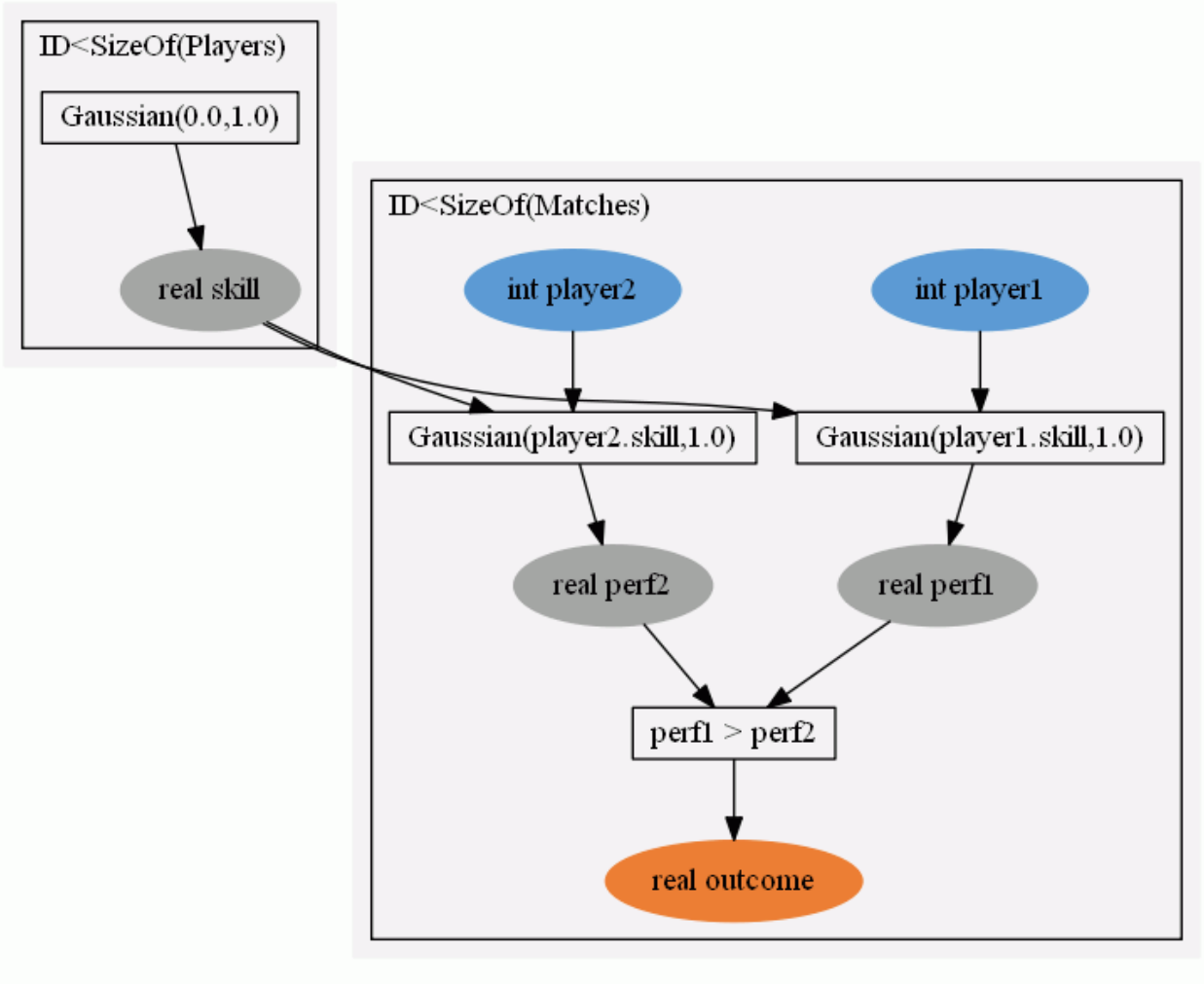
Tabular: A Schema-Driven Probabilistic Programming Language

Andy Gordon (MSR Cambridge and University of Edinburgh)

Based on joint work with Thore Graepel, Nicolas Rolland,
Claudio Russo, Johannes Borgström, John Guiver (MSR Cambridge)

ACM Principles of Programming Languages, January 2014

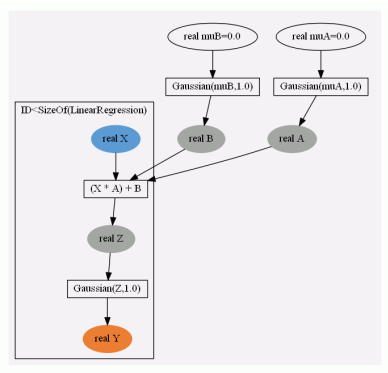
Graphical Models

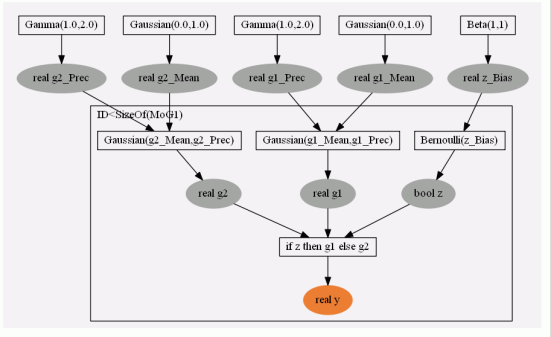


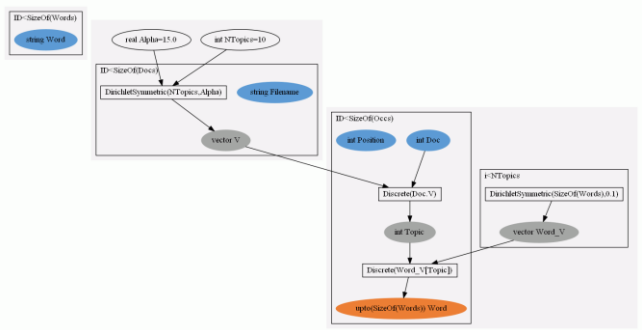
Sample

Infer

Graphical Models are Amazing!







- A **regression function** inputs a tuple of variables, and outputs one or more continuous variables.
- A **cluster analysis** groups items so that items in each cluster are more like each other than to items in other clusters.
- A **topic model** discovers the underlying topics so as to organise a set of documents.
- And many other models and variations…

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 graphical models from scratch.
- Several academic and commercial systems: BUGS, IBAL, Church, Dimple, STAN, **Infer.NET/Fun**, Factorie, BLOG, Alchemy and more.
- Actually, graphical models are pretty simple programs.

- What if we start from the schema of the observed data?

Aims of Tabular

- **DSL to get insight from data with graphical models**
- Empower the well-populated end-user side of the spectrum:
  - Data enthusiasts | ML devs | data scientists | ML PhDs
- 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

- POPL paper about 1 and 2, and shows how Tabular expresses InfernoDB, previous work on 3.

TrueSkill – Data

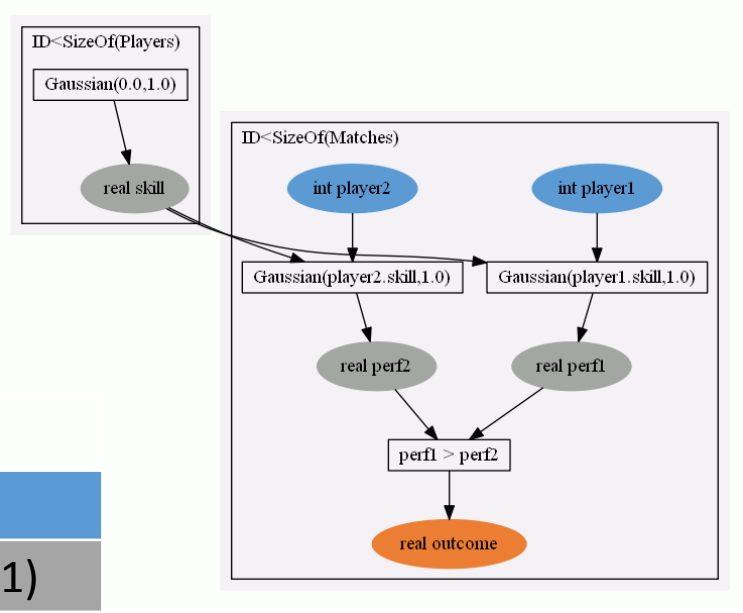
Players	
Name	String

Matches	
Player1	Link(Players)
Player2	Link(Players)
Win1	Bool

	Name
0	Alice
1	Bob
2	Cynthia

	Player1	Player2	Win1
0	0	1	False
1	1	2	False

TrueSkill



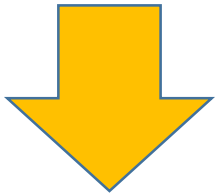
Players			
Name	string	input	
Skill	real	latent	Gaussian(25.0,0.01)

Matches			
Player1	link(Players)	input	
Player2	link(Players)	input	
Perf1	real	latent	Gaussian (Player1.Skill, 1.0)
Perf2	real	latent	Gaussian (Player2.Skill, 1.0)
Win1	bool	output	Perf1 > Perf2

Query-by-Latent Column

	Name
0	Alice
1	Bob
2	Cynthia

	Player1	Player2	Win1
0	0	1	False
1	1	2	False



	Skill
0	Gaussian(22.5,1.45)
1	Gaussian(25.22,1.53
2	Gaussian(27.93,1.45)

	Perf1	Perf2
0	Gaussian(22.5,1.1)	Gaussian(25.2,1.1)
1	Gaussian(25.2,1.1)	Gaussian(27.9,1.1)

Query-by-Missing-Value

	Name
0	Alice
1	Bob
2	Cynthia

	Player1	Player2	Win1
0	0	1	False
1	1	2	False
2	2	0	

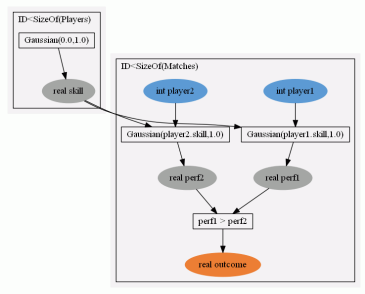
	Player1	Player2	Win1
0	0	1	False
1	1	2	False
2	2	0	Bernoulli(0.85)

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

Semantics of Schemas



Players			
Name	string	input	
Skill	real	latent	Gaussian(25.0,0.01)

For each row r of Players,
  Skill ~ Gaussian(25.0,0.01)

Matches			
Player1	link(Players)	input	
Player2	link(Players)	input	
Perf1	real	latent	Gaussian (Player1.Skill, 1.0)
Perf2	real	latent	Gaussian (Player2.Skill, 1.0)
Win1	bool	output	Perf1 > Perf2

For each row r of Matches,
  Player1 := r.Player1
  Player2 := r.Player2
  Perf1 ~ Gaussian(Players[Player1].Skill, 1.0)
  Perf2 ~ Gaussian(Players[Player2].Skill, 1.0)
  Win1 := (Perf1 > Perf2)
  if (r.Win1 not null)
   observe (Win1 == r.Win1)

Case study: Psychometrics

```
public static void CreateAndRunDAREModel(
    Gaussian abilityPrior, Gaussian difficultyPrior, Gamma discriminationPrior,
    int nParticipants, int nQuestions, int nChoices,
    int[] participantOfResponse, int[] questionOfResponse,
    int[] response, int[] trainingResponseIndices,
    int[] answer, int[] trainingQuestionIndices)
{  // Model
    var Evidence = Variable.Bernoulli(0.5).Named("evidence");
    var EvidenceBlock = Variable.If(Evidence);
    var NQuestions = Variable.New<int>().Named("nQuestions");
    var NParticipants = Variable.New<int>().Named("nParticipants");
    var NChoices = Variable.New<int>().Named("nChoices");
    var NResponses = Variable.New<int>().Named("nResponses");
    var NTrainingResponses = Variable.New<int>().Named("nTrainingResponses");
    var NTrainingQuestions = Variable.New<int>().Named("nTrainingQuestions");
    var p = new Range(NParticipants).Named("p");
    var q = new Range(NQuestions).Named("q");
    var c = new Range(NChoices).Named("c");
    var n = new Range(NResponses).Named("n");
    var tr = new Range(NTrainingResponses).Named("tr");
    var tq = new Range(NTrainingQuestions).Named("tq");
    var AnswerOfQuestion = Variable.Array<int>(q).Named("answer");
    AnswerOfQuestion[q] = Variable.DiscreteUniform(c).ForEach(q);
    var QuestionOfResponse = Variable.Array<int>(n).Named("questionOfResponse");
    var ParticipantOfResponse = Variable.Array<int>(n).Named("participantOfResponse");
    var TrainingQuestionIndices = Variable.Array<int>(tq).Named("trainingQuestionIndices");
    var TrainingResponseIndices = Variable.Array<int>(tr).Named("trainingResponseIndices");
    var ObservedResponseAnswer = Variable.Array<int>(tr).Named("observedResponseAnswer");
    var ObservedQuestionAnswer = Variable.Array<int>(tq).Named("observedQuestionAnswer");
    NTrainingQuestions.ObservedValue = nTrainingQuestions;
    ParticipantOfResponse.ObservedValue = participantOfResponse;
    QuestionOfResponse.ObservedValue = questionOfResponse;
    TrainingResponseIndices.ObservedValue = trainingResponseIndices;
    ObservedResponseAnswer.ObservedValue = Util.ArrayInit(nTrainingResponses, i =>
                        response[trainingResponseIndices[i]]);
    TrainingQuestionIndices.ObservedValue = trainingQuestionIndices;
    ObservedQuestionAnswer.ObservedValue = Util.ArrayInit(nTrainingQuestions, i =>
                        answer[trainingQuestionIndices[i]]);
    var Engine = new InferenceEngine()
    { ShowTimings = true,ShowWarnings = false,ShowProgress = false,NumberOfIterations = 10 };
    var AnswerOfResponsePosterior = Engine.Infer<Discrete[]>(AnswerOfResponse);
    var AnswerOfQuestionPosterior = Engine.Infer<Discrete[]>(AnswerOfQuestion);
    var LogEvidence = Engine.Infer<Bernoulli>(Evidence).LogOdds;
    var AbilityPosterior = Engine.Infer<Gaussian[]>(Ability);
    var DifficultyPosterior = Engine.Infer<Gaussian[]>(Difficulty);
    var DiscriminationPosterior = Engine.Infer<Gamma[]>(Discrimination);
}
```

Participants

Model	Lang.	LOC Data	LOC Model	LOC Inference	LOC Total	Compile (s)	Infer (s)	Model log evidence	Avg. log prob. test Responses	Avg. log prob. test answers
A	Tabular II	0	17	0	17	0.39	0.40	-7499.74	-1.432	-3.424
A	Infer.NET	73	45	20	138	0.32	0.38	-7499.74	-1.432	-3.425
DA	Tabular II	0	18	0	18	0.39	0.46	-5933.52	-1.118	-0.739
DA	Infer.NET	73	47	21	141	0.34	0.43	-5933.25	-1.118	-0.724
DARE	Tabular II	0	19	0	19	0.40	2.86	-5820.40	-1.119	-0.528
DARE	Infer.NET	73	49	22	144	0.37	2.8	-5820.40	-1.119	-0.528

Know	Bool	Latent	Probit(Advantage,QuestionID.Discrimination)
Guess	Int	Latent	DiscreteUniform(8)
Response	Int	Output	if Know then QuestionID.Answer else Guess

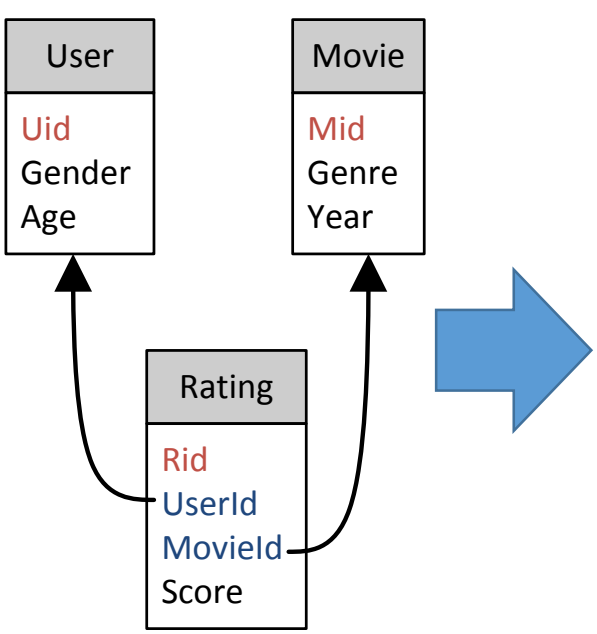
- Replicated Infer.NET C# by Bachrach et al 2012
- Same statistical results, much less code, slight loss in perf

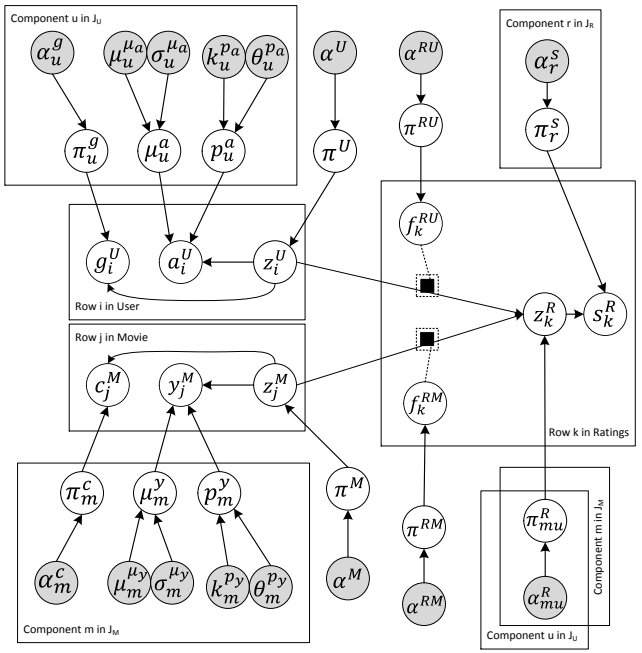
What else is in the paper?

schema	$S ::= (t_1\ T_1) \ldots (t_n\ T_n)$
table	$T ::= (c_1\ ty_1\ A_1) \ldots (c_n\ ty_n\ A_n)$
annotation	$A ::=$ **hyper**(E) | **param**(M) | **input** | **latent**(M) | **output**(M)
model expression	$M ::= E \mid P(h_1=E_1,\ldots,h_n=E_n) \mid M[E_{index}<E_{size}]$
probabilistic expression	$E ::= c \mid E.c \mid \mathrm{f}(E_1,\ldots,E_n) \mid \mathrm{D}(E_1,\ldots,E_n) \mid \ldots$
primitive model	$P ::=$ CBernoulli | CGaussian | CDiscrete | …

- Syntax; type system; semantics by translation to Fun.
- Theorem 1: the semantics respects the Tabular type system.
- Theorem 2: certain factor graph derived from the translation to Fun, correctly implements **query-by-latent-column**.
- Theorem 3: a transformation on Tabular schemas implements **query-by-missing-value** in terms of query-by-latent-column.

Automatic Model Suggestion

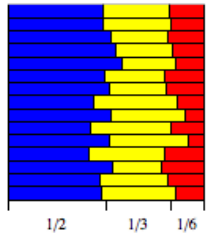




Automatically get recommender from relational schema

Problem: visual notation becoming unwieldy

Source: Wikipedia



Primitive Models (cf POPL'13)

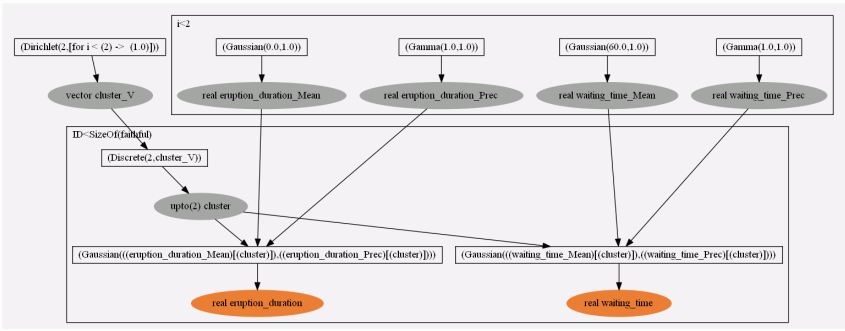
- A **Bayesian model** is a probabilistic function from **inputs** x to **outputs** y, governed by a **parameter** w, and fixed hyperparameter h.
- Model specified by a *Prior-part* and a *Gen-part*.
- We provide a range of distributions as models, packaged with their conjugate priors.
- The prior-part is used outside the loop for a table, the gen-part within the loop.

CDiscrete	
Default h	{N=3; alpha=1.0}
Prior(h)	Dirichlet(h.N,h.alpha)
Gen(h,w,x)	Discrete(w)

Ratings			
UserID	link(Users)	Input	
MovieID	link(Movies)	Input	
Score	int	Output	CDiscrete(N=5)

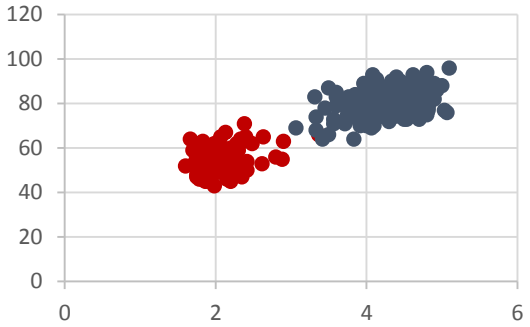
```
Score$ ~ Dirichlet(5,1.0)          // Prior
For each row r of Ratings,
  UserID := r.UserID
  MovieID:= r.MovieID
  Score ~ Discrete(Score$)         // Gen
  if (r.Score not null)
   observe (Score == r.Score)
```

Indexed Models

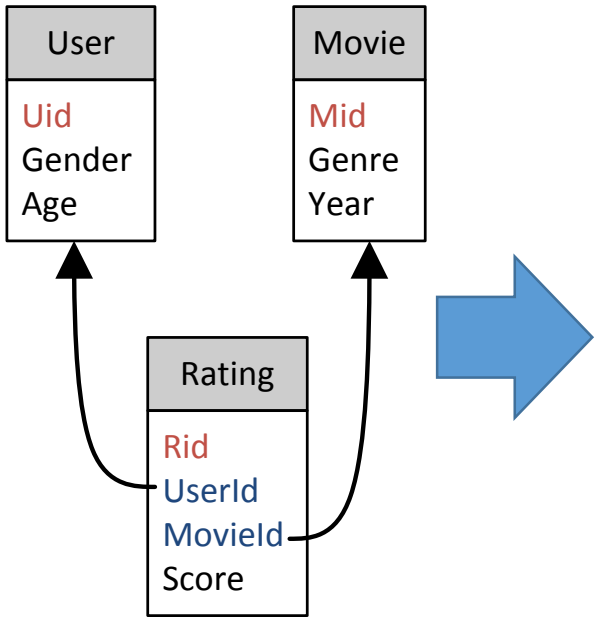


- An **indexed model** $M[E_{index}<E_{size}]$ is a model with an array of E_{size} copies of the parameter of M; the parameter to produce each output is selected by E_{index}.

Mixture			
cluster	int	latent	CDiscrete(N=2)
eruption_duration	real	output	CGaussian()[cluster<2]
waiting_time	real	output	CGaussian()[cluster<2]



Automatic Model Suggestion



Users			
UserCluster	Int	Latent	CDiscrete(N=4)
Gender	Bool	Output	CBernoulli()[UserCluster]
Age	Int	Output	CDiscrete(N=4)[UserCluster]

Movies			
MovieCluster	Int	Latent	CDiscrete(N=4)
Genre	Int	Output	CDiscrete(N=20)[MovieCluster]
Year	Int	Output	CDiscrete(N=100)[MovieCluster]

Ratings			
UserID	link(Users)	Input	
MovieID	link(Movies)	Input	
Score	int	Output	CDiscrete(N=5) [UserID.UserCluster] [MovieID.MovieCluster]

Model expressions allow us to recast early work on model suggestion as **probabilistic metaprogramming**

Tabular Contributions

- Programs are probabilistic annotations on existing schemas.
- Columns marked as **hyper**, **param**, **input**, **output**, or **latent** so as to define factor graphs with plates and parameters.
- **Query-by-latent-column** and **query-by-missing-value.**
- A grammar of model expressions to assemble complex hierarchical factor graphs:

$$M ::= E \mid P(h_1=E_1,\ldots,h_n=E_n) \mid M[E_{index}<E_{size}]$$

- Future work: functions and dependent types, time series, automatic suggestion of models from schemas

http://research.microsoft.com/fun