Probabilistic Programs as Spreadsheet Queries

Citation for published version:
Gordon, AD, Russo, CV, Szymczak, M, Borgström, J, Rolland, N, Graepel, T & Tarlow, D 2015, Probabilistic Programs as Spreadsheet Queries. in Programming Languages and Systems: 24th European Symposium on Programming, ESOP 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings. vol. 9032, Springer Berlin Heidelberg, pp. 1-25. DOI: 10.1007/978-3-662-46669-8_1

Digital Object Identifier (DOI):
10.1007/978-3-662-46669-8_1

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Programming Languages and Systems

Probabilistic Programs as Spreadsheet Queries

Andrew D. Gordon
Claudio Russo
Marcin Szymczak
Johannes Borgström
Nicolas Rolland
Thore Graepel
Daniel Tarlow

November 2014

Technical Report
MSR–TR–2014–135

Microsoft Research
21 Station Road
Cambridge, CB1 2FB
United Kingdom

Probabilistic Programs as Spreadsheet Queries

Andrew D. Gordon[1,2] Claudio Russo[1] Marcin Szymczak[2]
Johannes Borgström[3] Nicolas Rolland[1] Thore Graepel[1] Daniel Tarlow[1]

[1]Microsoft Research [2]University of Edinburgh [3]Uppsala University

Abstract. We describe the design, semantics, and implementation of a probabilistic programming language where programs are spreadsheet queries. Given an input database consisting of tables held in a spreadsheet, a query constructs a probabilistic model conditioned by the spreadsheet data, and returns an output database determined by inference. This work extends probabilistic programming systems in three novel aspects: (1) embedding in spreadsheets, (2) dependently-typed functions, and (3) typed distinction between random- and query-variables. It empowers users with knowledge of statistical modelling to do inference simply by editing textual annotations within their spreadsheets, with no other coding.

1 Spreadsheets and Typeful Probabilistic Programming

Probabilistic programming systems [14, 17] enable a developer to write a short piece of code that models a dataset, and then to rely on a compiler to produce efficient inference code to learn parameters of the model and to make predictions. Still, a great many of the world's datasets are held in spreadsheets, and accessed by users who are not developers. How can spreadsheet users reap the benefits of probabilistic programming systems?

Our first motivation here is to describe an answer, based on an overhaul of Tabular [16], a probabilistic language based on annotating the schema of a relational database. The original Tabular is a standalone application that runs fixed queries on a relational database (Microsoft Access). We began the present work by re-implementing Tabular within Microsoft Excel, with the data and program held in spreadsheets.

The conventional view is that the purpose of a probabilistic program is to define the random-variables whose marginals are to be determined (as in the query-by-missing-value of original Tabular). In our experience with spreadsheets, we initially took this view, and relied on Excel formulas, separate from the probabilistic program, for post-processing tasks such as computing the mode (most likely value) of a distribution, or deciding on an action (whether or not to place a bet, say). We found, to our surprise, that combining Tabular models and Excel formulas is error-prone and cumbersome, particularly when the sizes of tables changes, the parameters of the model change, or we simply need to update a formula for every row of a column.

In response, our new design contributes the principle that a probabilistic program defines a pseudo-deterministic query on data. The query is specified in terms of three sorts of variable: (1) deterministic variables holding concrete input data; (2) nondeterministic random-variables constituting the probabilistic model conditioned on input data; and (3) pseudo-deterministic query-variables defining the result of the program (instead of using Excel formulas). Random-variables are defined by draws from a set of builtin distributions. Query-variables are defined via an **infer** primitive that returns the marginal posterior distributions of random-variables. For instance, given a random-variable of Boolean type, **infer** returns the probability p that the variable is **true**. In theory, **infer** is deterministic—it has an exact semantics in terms of measure theory; in practice, **infer** (and hence the whole query) is only pseudo-deterministic, as implementations almost always perform approximate or nondeterministic inference. We have many queries as evidence that post-processing can be incorporated into the language.

Our second motivation is to make a case for *typeful probabilistic programming* in general, with evidence from our experience of overhauling Tabular for spreadsheets. Cardelli [7] identifies the programming style based on widespread use of mechanically-checked types as *typeful programming*. Probabilistic languages that are embedded DSLs, such as HANSEI [19], Fun [3], and Factorie [22], are already typeful in that they inherit types from their host languages, while standalone languages, such as BUGS [11] or Stan [35], have value-indexed data schemas (but no user-defined functions). Still, we find that more sophisticated forms of type are useful in probabilistic modelling.

We make two general contributions to typeful probabilistic programming.

(1) *Value-indexed function types usefully organise user-defined components, such as conjugate pairs, in probabilistic programming languages.*

We allow value indexes in types to indicate the sizes of integer ranges and of array dimensions. We add value-indexed function types for user-defined functions, with a grid-based syntax. The paper has examples of user-defined functions (such as Action in Section 6) showing their utility beyond the fixed repertoire of conjugate pairs in the original Tabular. An important difficulty is to find a syntax for functions and their types that fits with the grid-based paradigm of spreadsheets.

(2) *A type-based information-flow analysis usefully distinguishes the stochastic and deterministic parts of a probabilistic program.*

To track the three sorts of variable, each type belongs to a *space* indicating whether it is: (**det**) deterministic input data, (**rnd**) a non-deterministic random-variable defining the probabilistic model of the data, or (**qry**) a pseudo-deterministic query-variable defining a program result. Spaces allow a single language to define both model and query, while the type system governs flows between the spaces: data flows from **rnd** to **qry** via **infer**, but to ensure that a query needs only a single run of probabilistic inference, there are no flows from **qry** to **rnd**. There is an analogy between our spaces and levels in information flow systems: **det**-space is like a level of trusted data; **rnd**-space is like a level of untrusted data that is tainted by randomness; and **qry** is like a level of trusted data that includes untrusted data explicitly endorsed by **infer**.

The benefits of spaces include: (1) to document the role of variables, (2) to slice a program into the probabilistic model versus the result query, and (3) to prevent accidental errors. For instance, only variables in **det**-space may appear as indexes in types to guarantee that our models can be compiled to the finite factor-graphs supported by inference backends such as Infer.NET [23].

This paper defines the syntax, semantics, and implementation of a new, more typeful Tabular. Our implementation is a downloadable add-in for Excel. For execution on data in a spreadsheet, a Tabular program is sliced into (1) an Infer.NET model for inference, and (2) a C# program to compute the results to be returned to the spreadsheet.

The original semantics of Tabular uses the higher-order model-learner pattern [15], based on a separate metalanguage. Given a Tabular schema $\mathbb{S}$ and an input database DB that matches $\mathbb{S}$, our semantics consists of two algorithms.

(1) An algorithm $\mathsf{CoreSchema}(\mathbb{S})$ applies a set of source-to-source reductions on $\mathbb{S}$ to yield $\mathbb{S}'$, which is in a core form of Tabular without user-defined functions and some other features.
(2) An algorithm $\mathsf{CoreQuery}(\mathbb{S}', DB)$ first constructs a probabilistic model based on the **rnd**-space variables in $\mathbb{S}'$ conditioned by DB, and then evaluates the **qry**-space variables in $\mathbb{S}'$ to assemble an output database DB'.

Our main technical results about the semantics are as follows.

(1) Theorem 1 establishes that $\mathsf{CoreSchema}(\mathbb{S})$ yields the unique core form $\mathbb{S}'$ of a well-typed schema $\mathbb{S}$, as a corollary of standard properties of our reduction relation with respect to the type system (Proposition 1, Proposition 2, and Proposition 3).
(2) Theorem 2 establishes pre- and post-conditions of the input and output databases when $DB' = \mathsf{CoreQuery}(\mathbb{S}', DB)$.

Beyond theory, the paper describes many examples of the new typeful features of Tabular, including a detailed account of Bayesian Decision Theory, an important application of probabilistic programming, not possible in the original form of Tabular. A language like IBAL or Figaro allows for rational decision-making, but via decision-specific language features, rather than in the core expression language. We present a numeric comparison of a decision theory problem expressed in Tabular versus the same problem expressed in C# with direct calls to Infer.NET, showing that we pay very little in performance in return for a much more succinct spreadsheet program. As evidence that Tabular is widely applicable an appendix lists over a dozen different models, evaluated on millions of rows of data, that are available in our download.

Structure of the Paper Section 2 motivates Tabular's new syntax for functions and queries with a worked example.

Section 3 presents our new design for Tabular, including its deterministic reduction relation to transform to core form, without function calls or indexed models.

Section 4 introduces a source-to-source transformation that reduces programs to a core form.

Section 5 introduces our system of dependent types, and formulates the algorithms CoreSchema and CoreQuery, and states their correctness as Theorem 1 and Theorem 2.

Section 6 recalls the standard Bayesian framework for making decisions under uncertainty, and shows how such decisions may be expressed in Tabular using **rnd**-space variables for the probabilistic model, and **qry**-space variables to compute decisions.

Section 7 describes our implementation, including evaluations. Section 8 discusses related work, and Section 9 concludes.

Additional details and proofs omitted from the main body of the paper are grouped in an appendix.

Appendix A lists some builtin functions. Appendix B describes over a dozen example models run using Tabular in Excel. Appendix C shows the C# code generated from the Old Faithful clustering model.

Appendix D defines alpha-conversion on the syntax of Tabular, and shows some syntactic properties of the reduction relations. Appendix E defines the semantics of Core Tabular in detail, thus completing the sketch of the semantics in Section 5.3. Appendix F gives proofs for Proposition 1, Proposition 2, and Proposition 3 stated in Section 5.

2 Functions and Queries, by Example

Primer: Discrete and Dirichlet Distributions To begin to describe the new features of Tabular, we recall a couple of standard distributions. If array $\mathsf{V} = [p_0;\dots;p_{n-1}]$ is a *probability vector* (that is, each p_i is a probability and they sum to 1) then $\mathsf{Discrete}[n](\mathsf{V})$ is the discrete distribution that yields a sample $i \in 0..n-1$ with probability p_i. The distribution $\mathsf{Discrete}[2]([\frac{1}{2};\frac{1}{2}])$ models a coin that we know to be fair. If we are uncertain whether the coin is fair, we need a distribution on probability vectors to represent our uncertainty. The distribution $\mathsf{Dirichlet}[n]([c_0;\dots;c_{n-1}])$ on a probability vector V represents our uncertainty after observing a count $c_i - 1$ of samples of i from $\mathsf{Discrete}[n](\mathsf{V})$ for $i \in 0..n-1$. We omit the formal definition, but discuss the case $n = 2$.

A probability vector V drawn from $\mathsf{Dirichlet}[2]([t+1;h+1])$ represents our uncertainty about the bias of a coin after observing t tails and h heads. It follows that $\mathsf{V} = [1-p;p]$ where p is the probability of heads. The expected value of p is $\frac{h+1}{t+h+2}$, and the variance of p diminishes as t and h increase. If $t = h = 0$, the expected value is $\frac{1}{2}$ and p is uniformly distributed on the unit interval. If $t = h = 10$ say, the expected value remains $\frac{1}{2}$ but p is much more likely near the middle than the ends of the interval.

Review: Probabilistic Schemas in Tabular Suppose we have a table named Coins with a column Flip containing a series of coin flips and wish to infer the bias of the coin. (The syntax [**for** i $< 2 \rightarrow 1.0$] is an array comprehension, in this case returning $[1.0, 1.0]$.)

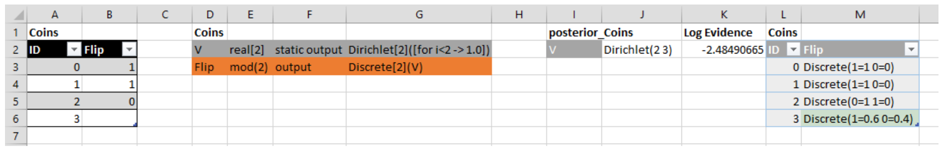
Coins

ID	Flip
0	1
1	1
2	0

table Coins (original Tabular)			
V	**real**[]	**static output**	Dirichlet[2]([**for** i < 2 →1.0])
Flip	**int**	**output**	Discrete[2](V)

The model above (in original Tabular up to keyword renaming) is read as a probabilistic recipe for generating the coin flips from the unknown parameter V, conditioned on the actual dataset. The first line creates a random variable $\mathsf{V} = [1-p;p]$ from $\mathsf{Dirichlet}[2]([1;1])$, which amounts to choosing the probability p of heads uniformly from the unit interval. The second line creates a random variable Flip from $\mathsf{Discrete}[n](V)$ for each row of the table and conditions the variable in each row to equal the actual observed coin flip, if it is present. Each Tabular variable is either at **static**- or **inst**-level. A **static**-variable occurs just once per table, whereas an **inst**-variable occurs for each row of the table. The default level is **inst**, so Flip is at **inst**-level.

Now, suppose the data for the column Flip is $[1;1;0]$; the prior distribution of V is updated by observing 2 heads and 1 tails, to yield the posterior Dirichlet$[2]([2;3])$, which has mean $\frac{3}{5}$. Given our example model, the fixed queries of this initial form of Tabular compute the posterior distribution of V, and write the resulting distributions as strings into the spreadsheet, as shown below. The missing value in cell B6 of the Flip column is predicted by the distribution in cell M6: 60% chance of 1, 40% chance of 0. (Cells E2 and E3 show dependent types of our new design, not of the original Tabular.)



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Coins			Coins					posterior_Coins		Log Evidence	Coins	
2	ID	Flip		V	real[2]	static output	Dirichlet[2]([for i<2 -> 1.0])		V	Dirichlet(2 3)	-2.48490665	ID	Flip
3	0	1		Flip	mod(2)	output	Discrete[2](V)					0	Discrete(1=1 0=0)
4	1	1										1	Discrete(1=1 0=0)
5	2	0										2	Discrete(0=1 1=0)
6	3											3	Discrete(1=0.6 0=0.4)
7													

New Features of Tabular Our initial experience with the re-implementation shows that writing probabilistic programs in spreadsheets is viable but suggests three new language requirements, explained in the remainder of this section.

(1) User-defined functions for abstraction (to generalize the fixed repertoire of primitive models in the original design).
(2) User-defined queries to control how parameters and predictions are inferred from the model and returned as results (rather than simply dumping raw strings from fixed queries).
(3) Value-indexed dependent types (to catch errors with vectors, matrices, and integer ranges, and help with compilation).

Next, we sketch how our new design meets these requirements.

(1) User-Defined Functions The Coins example shows the common pattern of a discrete distribution with a Dirichlet prior. We propose to write a function for such a pattern as follows. It explicitly returns the ret output but also implicitly returns the V output.

fun CDiscrete			
N	**int!det**	**static input**	
R	**real!rnd**	**static input**	
V	**real**[N]**!rnd**	**static output**	Dirichlet[N]([**for** $i < N \to$ R])
ret	**mod**(N)**!rnd**	**output**	Discrete[N](V)

In a table description, **input**-attributes refer implicitly to fully-observed columns in the input database. On the other hand, a function is explicitly invoked using syntax like CDiscrete$(N = 2; R = 1)$, and the **input**-attributes N and R refer to the argument expressions, passed call-by-value.

(2) User-Defined Queries To support both the construction of probabilistic models for inference, and the querying of results, we label each type with one of three *spaces*:

(1) **det**-space is for fully-observed input data;
(2) **rnd**-space is for probabilistic models, conditioned by partially-observed input data;

(3) **qry**-space is for deterministic results queried from the inferred marginal distributions of **rnd**-space variables.

We organise the three spaces via the least partial order given by **det** $<$ **rnd** and **det** $<$ **qry**, so as to induce a subtype relation on types. Moreover, to allow flows from **rnd**-space to **qry**-space, an operator **infer**.$D.y_i(x)$ computes the parameter y_i in **qry**-space of the marginal distribution $D(y_1,\ldots,y_n)$ of an input x in **rnd**-space.

For example, here is a new model of our Coins table, using a call to CDiscrete to model the coin flips in **rnd**-space, and to implicitly define a **rnd**-space variable V for the bias of the coin. Assuming our model is conditioned by data $[1;1;0]$, the marginal distribution of V is Dirichlet[2]([2;3]) where [2;3] is the counts-parameter. Hence, the call **infer**.Dirichlet[2].counts(V) yields [2;3], and the query returns the mean $\frac{3}{5}$.

table Coins			
Flip	**mod**(2)!**rnd**	**output**	CDiscrete(N=2;R=1)*(∗returns Flip and Flip_V∗)*
counts	**real**[2]!**qry**	**static local**	**infer**.Dirichlet[2].counts(Flip_V)
Mean	**real**!**qry**	**static output**	counts[1]/(counts[1]+counts[0])

Our reduction relation rewrites this schema to the following core form.

table Coins			
R	**real**!**rnd**	**static local**	1
Flip_V	**real**[2]!**rnd**	**static output**	Dirichlet[2]([**for** i < 2 → R])
Flip	**mod**(2)!**rnd**	**output**	Discrete[2](V)
counts	**real**[2]!**qry**	**static local**	**infer**.Dirichlet[2].counts(Flip_V)
Mean	**real**!**qry**	**static output**	counts[1]/(counts[1]+counts[0])

(3) Simple Dependent Types Our code has illustrated dependent types of statically-sized arrays and integer ranges: values of $T[e]$ are arrays of T of size e, while values of **mod**(e) are integers in the set $0..(e-1)$. In both cases, the size e must be a **det**-space **int**. (Hence, the dependence of types on expressions is simple, and all sizes may be resolved statically, given the sizes of tables.) The use of dependent types for arrays is standard (as in Dependent ML [38]); the main subtlety in our probabilitic setting is the need for spaces to ensure that indexes are deterministic.

Primitive distributions have dependent types:

Distributions: $D_{spc} : [x_1 : T_1,\ldots,x_m : T_m](y_1 : U_1,\ldots,y_n : U_n) \to T$

Discrete$_{spc}$: [N : **int**!**det**](probs : **real**!spc[N]) → **mod**(N)!**rnd**
Dirichlet$_{spc}$: [N : **int**!**det**](counts : **real**!spc[N]) → (**real**!**rnd**)[N]

User-defined functions have dependent types written as grids, such as the following type $Q_{CDiscrete}$ for CDiscrete:

N	**int**!**det**	**static input**
R	**real**!**rnd**	**static input**
V	**real**[N]!**rnd**	**static output**
ret	**mod**(N)!**rnd**	**output**

Finally, the table type for our whole model of the Coins table is the following grid. It lists the **rnd**-space variables returned by CDiscrete as well as the explicitly defined Mean. Attributes marked as **local** are private to a model or function, are identified up to alpha-equivalence, and do not appear in types. Attributes marked as **input** or **output** are binders, but are not identified up to alpha-equivalence, and are exported from tables or functions. Their names must stay fixed because of references from other tables.

V	**real**[2]**!rnd**	**static output**
Flip	**mod**(2)**!rnd**	**output**
Mean	**real!qry**	**static output**

3 Syntax of Tabular Enhanced with Functions and Queries

We describe the formal details of our revision of Tabular in this section. In the next, Section 4, we show how features such as function applications may be eliminated by reducing schemas to a core form with a direct semantics.

Column-Oriented Databases Let t range over table names and c range over attribute names. We consider a database to be a pair $DB = (\delta_{in}, \rho_{sz})$ consisting of a record of tables $\delta_{in} = [t_i \mapsto \tau_i \,^{i \in 1..n}]$, and a valuation $\rho_{sz} = [t_i \mapsto sz_i \,^{i \in 1..n}]$ holding the number of rows $sz_i \in \mathbb{N}$ in each column of table t_i. Each table $\tau_i = [c_i \mapsto a_i \,^{j \in 1..m_i}]$ is a record of attributes a_i. An *attribute* is a value V tagged with a *level* ℓ. An attribute is normally a whole *column* $\mathsf{inst}(V)$, where V is an array of length sz_i and the level **inst** is short for "instance". It may also be a single value, $\mathsf{static}(V)$, a *static attribute*. The main purpose of allowing static attributes is to return individual results (such as Mean in our Coins example) from queries. The static attributes of a Tabular table can be implemented in a relational store by introducing an auxiliary relation with a single row containing the static attributes.

Databases, Tables, Attributes, and Values:

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

For example, the data for our Coins example is $DB = (\delta_{in}, \rho_{sz})$ where $\delta_{in} = [\mathsf{Coins} \mapsto [\mathsf{Flip} \mapsto \mathsf{inst}([1;1;0])]]$ and $\rho_{sz} = [\mathsf{Coins} \mapsto 3]$.

In examples, we assume each table has an implicit primary key ID and that the keys are in the range $0..sz_i - 1$. A value V may contain occurrences of "?", signifying missing data; we write known(V) if V contains no occurrence of ?. Otherwise, a value may be an array, or a constant s: either a Boolean, integer, or real.

Syntax of Tabular Expressions and Schemas An *index expression e* may be a variable x or a constant, and may occur in types (as the size of an array, for instance). Given a database $DB = (\delta_{in}, \rho_{sz})$, $\mathsf{sizeof}(t)$ denotes the constant $\rho_{sz}(t)$. Attribute names c (but not table names) may occur in index expressions as variables. A *attribute type T* can be a scalar, a bounded non-negative integer or an array. Each type has an associated *space* (which is akin to an information-flow level, but independent of the notion of level in Tabular, introduced later on). (The type system is discussed in detail in Section 5.)

Index Expressions, Spaces and Dependent Types of Tabular:

$e ::=$	index expression
$\quad x$	variable
$\quad s$	scalar constant
$\quad \mathsf{\textbf{sizeof}}(t)$	size of a table
$S ::= \mathsf{\textbf{bool}} \mid \mathsf{\textbf{int}} \mid \mathsf{\textbf{real}}$	scalar type
$spc ::= \mathsf{\textbf{det}} \mid \mathsf{\textbf{rnd}} \mid \mathsf{\textbf{qry}}$	space
$T,U ::= (S \,!\, spc) \mid (\mathsf{\textbf{mod}}(e) \,!\, spc) \mid T[e]$	(attribute) type

$\text{space}(S\,!\,spc) \triangleq spc \quad \text{space}(T[e]) \triangleq \text{space}(T) \quad \text{space}(\mathsf{\textbf{mod}}(e)\,!\,spc) \triangleq spc$
$spc(T) \triangleq \text{space}(T) = spc$

We write $\mathsf{\textbf{link}}(t)$ as a shorthand for $\mathsf{\textbf{mod}}(\mathsf{\textbf{sizeof}}(t))$, for foreign keys to table t.

The syntax of (full) *expressions* includes index expressions, plus deterministic and random operations. We assume sets of deterministic functions g, and primitive distributions D. These have type signatures, as illustrated for Discrete and Dirichlet in Section 2. In $D[e_1,\ldots,e_m](F_1,\ldots,F_n)$, the arguments e_1, …, e_m index the result type, while $F_1,\ldots,F_n$ are parameters to the distribution. The operator $\mathsf{\textbf{infer}}.D[e_1,\ldots,e_m].y(x)$ is described intuitively in Section 2. We write $\text{fv}(T)$ and $\text{fv}(E)$ for the sets of variables occurring free in type T and expression E.

Expressions of Tabular:

$E,F ::=$	expression
$\quad e$	index expression
$\quad g(E_1,\ldots,E_n)$	deterministic primitive g
$\quad D[e_1,\ldots,e_m](F_1,\ldots,F_n)$	random draw from distribution D
$\quad \mathsf{\textbf{if}}\ E\ \mathsf{\textbf{then}}\ F_1\ \mathsf{\textbf{else}}\ F_2$	if-then-else
$\quad [E_1,\ldots,E_n] \mid E[F]$	array literal, lookup
$\quad [\mathsf{\textbf{for}}\ x < e \rightarrow F]$	for-loop (scope of index x is F)
$\quad \mathsf{\textbf{infer}}.D[e_1,\ldots,e_m].y(x)$	parameter y of inferred marginal of x
$\quad E : t.c$	dereference link E to instance of c
$\quad t.c$	dereference static attribute c of t

A Tabular schema is a relational schema with each attribute annotated not just with a type T, but also with a *level* ℓ, a *visibility viz*, and a *model expression M*.

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
$viz ::= \mathsf{\textbf{input}} \mid \mathsf{\textbf{local}} \mid \mathsf{\textbf{output}}$	visibility
$M,N ::= \varepsilon \mid E \mid M[e_{index} < e_{size}] \mid \mathbb{T}\ R$	model expression
$R ::= (c_1 = e_1,\ldots,c_n = e_n)$	function arguments

For $(c : T\ \ell\ viz\ M)$ to be well-formed, $viz =$ **input** if and only if $M = \varepsilon$. We only consider well-formed declarations. The visibility viz indicates whether the attribute c is given as an input, defined locally by the model expression M, or defined as an output by the model expression M. When omitted, the level of an attribute defaults to **inst**. (We define the formal relationship between a schema and its input and output databases in Appendix E.3.)

Functions, Models, and Model Expressions A challenge for this paper was to find a syntax for functions that is compatible with the grid-format of spreadsheets; we do so by re-interpreting the syntax $\mathbb{T}$ for tables as also the syntax of functions. A *function* is a table of the form $\mathbb{T} = [\mathsf{col}_1; \ldots; \mathsf{col}_n; (\mathsf{ret} : T\ \textbf{output}\ E)]$. A *model* is a function where each col_i is a local or an output. A *model expression M* denotes a *model* as follows:

- An expression E denotes the model that simply returns E.
- A *function application* $\mathbb{T}\ (c_1 = e_1, \ldots, c_n = e_n)$ denotes the function $\mathbb{T}$, but with each of its inputs c_i replaced by e_i.
- An *indexed model* $M[e_{index} < e_{size}]$ denotes the model for M, but with any **rnd static** attribute c replicated e_{size} times, as an array, and with references to c replaced by the lookup $c[e_{index}]$.

Formally, functions are embedded within our syntax of function applications $\mathbb{T}\ R$. In practice, our implementation supports separate function definitions written as **fun** $f\ \mathbb{T}$, such as CDiscrete in Section 1 and CG in Section 6. A function reference (within a model expression) is written $f\ R$ to stand for $\mathbb{T}\ R$.

Indexed models appear in the original Tabular, while function applications are new.

Binders and Alpha-Equivalence All attribute names c are considered bound by their declarations. The names of **local** attributes are identified up to alpha-equivalence. The names of **input** and **output** attributes are considered as fixed identifiers (like the fields of records) that export values from a table, and are not identified up to alpha-equivalence, because changing their names would break references to them.

Let $\text{inputs}(\mathbb{T})$ be the **input** attributes of table $\mathbb{T}$, that is, the names c in $(c : T\ \ell\ \textbf{input}\ \varepsilon)$. Let $\text{locals}(\mathbb{T})$ be all the **local** attributes of table $\mathbb{T}$, that is, the names c in $(c : T\ \ell\ \textbf{local}\ M)$. Let $\text{outputs}(\mathbb{T})$ be all the **output** attributes of table $\mathbb{T}$, that is, the names c in $(c : T\ \ell\ \textbf{output}\ M)$ plus $\text{outputs}(M)$, where the latter consists of the union of $\text{outputs}(\mathbb{T}_i)$ for any applications of $\mathbb{T}_i$ within M. Let $\text{dom}(\mathbb{T})$ be the union $\text{inputs}(\mathbb{T}) \cup \text{locals}(\mathbb{T}) \cup \text{outputs}(\mathbb{T})$. Hence, the free variables $\text{fv}(\mathbb{T})$ are given by:

$$\text{fv}((c : T\ \ell\ viz\ M) :: \mathbb{T}') \triangleq \text{fv}(T) \cup \text{fv}(M) \cup (\text{fv}(\mathbb{T}') \setminus (\{c\} \cup \text{outputs}(M))) \qquad \text{fv}([]) \triangleq \{\}$$

4 Reducing Schemas to Core Tabular

We define reduction relations that explain the meaning of function calls and indexed models by rewriting, and hence transforms any well-typed schema to a core form. The reduction semantics allows us to understand indexed models, and also function calls, within the Tabular syntax. Hence, this semantics is more direct and self-contained than the original semantics of Tabular [16], based on translating to a semantic metalanguage.

If all the attributes of a schema are simple expressions E instead of arbitrary model expressions, we say it is in *core* form:

Core Attributes, Tables, and Schemas:

$\text{Core}((c : T\ \ell\ \textbf{input}\ \varepsilon)) \quad \text{Core}((c : T\ \ell\ \textbf{local}\ E)) \quad \text{Core}((c : T\ \ell\ \textbf{output}\ E))$
$\text{Core}([\mathsf{col}_1; \ldots ; \mathsf{col}_n])$ if $\text{Core}(\mathsf{col}_1), \ldots, \text{Core}(\mathsf{col}_n)$
$\text{Core}([t_i = \mathbb{T}_i\ ^{i \in 1..n}])$ if $\text{Core}(\mathbb{T}_i)$ for each $i \in 1..n$

To help explain our reduction rules, consider the following function definition.

fun CG		
M	**real!det static input**	
P	**real!det static input**	
Mean	**real!rnd static output**	GaussianFromMeanAndPrecision(M,P)
Prec	**real!rnd static output**	Gamma(1.0,1.0)
ret	**real!rnd output**	GaussianFromMeanAndPrecision(Mean,Prec)

The following mixture model, for a dataset consisting of durations and waiting times for Old Faithful eruptions, uses three function applications and two indexed models. Each row of the model belongs to one of two clusters, indicated by the attribute cluster; the indexed models for duration and time give different means and precisions depending on the value of cluster. Since cluster is an **output**, Tabular allows missing values in that column (and indeed they are all missing), but the **qry**-space assignment returns the most likely cluster for each row as the result of the query.

table faithful			
cluster	**mod**(2)**!rnd**	**output**	CDiscrete(N=2)
duration	**real!rnd**	**output**	CG(M=0.0, P=1.0)[cluster < 2]
time	**real!rnd**	**output**	CG(M=60.0, P=1.0)[cluster < 2]
assignment	**mod**(2)**!qry**	**output**	ArgMax(**infer**.Discrete[2].probs(cluster))

To eliminate compound model expressions from schemas such as this, we define rules for the reduction of function applications and the operation of table indexing.

The relation $\mathbb{T} \vdash R \rightsquigarrow_o \mathbb{T}_1$ means that $\mathbb{T}_1$ is the outcome of substituting the arguments R for the **input** attributes of the function $\mathbb{T}$, within an attribute named o. For example, for the function application in the duration attribute, we have $\mathsf{CG} \vdash [M = 0.0, p = 1.0] \rightsquigarrow_{\mathsf{duration}} \mathsf{CG}_1$, where CG_1 is as follows:

duration_Mean	**real!rnd static output**	GaussianFromMeanAndPrecision(0.0, 1.0)
duration_Prec	**real!rnd static output**	Gamma(1.0, 1.0)
duration	**real!rnd output**	GaussianFromMeanAndPrecision(duration_Mean, duration_Prec)

The inductive definition follows. Rule (Apply Input) instantiates an **input** c with an argument e; (Apply Skip) prefixes **local** and **output** attributes with o; and (Apply Ret) turns the ret attribute of the function into name o of the call-site.

Inductive Definition of Function Application: $\mathbb{T} \vdash R \rightsquigarrow_o \mathbb{T}_1$

(Apply Ret)
$$\frac{}{[(\mathsf{ret} : T\ \ell\ viz\ E)] \vdash [] \rightsquigarrow_o [(o : T\ \ell\ viz\ E)]}$$

(Apply Input)
$$\frac{\mathbb{T}\{^e/_c\} \vdash R \rightsquigarrow_o \mathbb{T}_1 \quad \text{dom}(\mathbb{T}) \cap \text{fv}(e) = \varnothing}{(c : T\ \ell\ \textbf{input}\ \varepsilon) :: \mathbb{T} \vdash [c = e] :: R \rightsquigarrow_o \mathbb{T}_1}$$

(APPLY SKIP) ($viz \in \{\textsf{local}, \textsf{output}\}$)

$$\frac{\mathbb{T}\{^{o\_c}/_{c}\} \vdash R \leadsto_o \mathbb{T}_1 \qquad c \notin \mathrm{fv}(R)}{(c : T\ \ell\ viz\ E) :: \mathbb{T} \vdash R \leadsto_o (o\_c : T\ \ell\ viz\ E) :: \mathbb{T}_1}$$

Next, we define $\mathbb{T}[e_1 < e_2]$ to be the outcome of indexing the **static rnd** variables of a core table $\mathbb{T}$, that is, turning each into an array of size e_2, and references to those variables into array accesses indexed by e_1. For instance, CG_1[cluster $<$ 2] is equal to:

duration_Mean	**real**[2]!**rnd**	**static**	**output**	[**for** _ < 2 → GaussianFromMeanAndPrecision(0.0, 1.0)]
duration_Prec	**real**[2]!**rnd**	**static**	**output**	[**for** _ < 2 → Gamma(1.0, 1.0)]
duration	**real**!**rnd**		**output**	GaussianFromMeanAndPrecision(duration_Mean[cluster], duration_Prec[cluster])

Table Indexing: $\mathbb{T}[e_1 < e_2]$

$[\,][e_1 < e_2] \triangleq [\,]$
$((c : T\ \ell\ \textsf{input}\ \varepsilon) :: \mathbb{T})[e_1 < e_2] \triangleq ((c : T\ \ell\ \textsf{input}\ \varepsilon) :: \mathbb{T}[e_1 < e_2])$
$((c : T\ \ell\ viz\ E) :: \mathbb{T})[e_1 < e_2] \triangleq$
$\quad (c : T[e_2]\ \ell\ viz\ [\textsf{for}\ \_ < e_2 \rightarrow E]) :: \mathbb{T}\{^{c[e_1]}/_{c}\}[e_1 < e_2]$
$\qquad$ if $\ell = \textsf{static}$ and $viz \neq \textsf{input}$ and $\textsf{rnd}(T)$
$\quad (c : T\ \ell\ viz\ E) :: \mathbb{T}[e_1 < e_2]$ $\qquad$ if $\ell = \textsf{inst}$ or $viz = \textsf{input}$ or $\neg\textsf{rnd}(T)$

Below, we give inductive definitions of reduction relations on schemas, tables, and model expressions. There are congruence rules, plus (RED INDEX) and (RED INDEX EXPR) for indexed models, and (RED APPL) for applications. The latter needs additional operations $\mathbb{T} \wedge \ell$ and $\mathbb{T} \wedge viz$, to adjust the model $\mathbb{T}$ of function body to the level ℓ and visibility viz of the call-site. These operators drop any **output** attributes to **local**, if the callsite is **local**, and drop any **inst**-level attributes to **static**, if the callsite is **static**.

- Consider the 2-point lattice **static** $<$ **inst**. Let $\mathbb{T} \wedge \ell$ be the outcome of changing each $(c : T\ \ell_c\ viz\ M)$ in $\mathbb{T}$ to $(c : T\ (\ell_c \wedge \ell)\ viz\ M)$. Hence, $\mathbb{T} \wedge \textsf{inst}$ is the identity, while $\mathbb{T} \wedge \textsf{static}$ drops **inst** variables to **static** variables.
- Consider the 2-point lattice **local** $<$ **output**. Let $\mathbb{T} \wedge viz$ be the outcome of changing each $(c : T\ \ell\ viz_c\ M)$ in $\mathbb{T}$ to $(c : T\ \ell\ (viz_c \wedge viz)\ M)$. Hence, $\mathbb{T} \wedge \textsf{output}$ is the identity, while $\mathbb{T} \wedge \textsf{local}$ drops **output** variables to **local** variables.

Judgments:

$\mathbb{S} \rightarrow \mathbb{S}'$	schema reduction
$\mathbb{T} \rightarrow \mathbb{T}'$	table reduction
$M \rightarrow M'$	model reduction

Reduction Relations: $\mathbb{S} \rightarrow \mathbb{S}'$, $\mathbb{T} \rightarrow \mathbb{T}'$, $M \rightarrow M'$

(RED SCHEMA LEFT)

$$\frac{\mathbb{T} \rightarrow \mathbb{T}'}{(t = \mathbb{T}) :: \mathbb{S} \rightarrow (t = \mathbb{T}') :: \mathbb{S}}$$

(RED SCHEMA RIGHT)

$$\frac{\mathbb{S} \rightarrow \mathbb{S}' \quad \mathsf{Core}(\mathbb{T})}{(t = \mathbb{T}) :: \mathbb{S} \rightarrow (t = \mathbb{T}) :: \mathbb{S}'}$$

$$\text{(RED MODEL)}\quad \frac{M \to M'}{(c : T\ \ell\ viz\ M) :: \mathbb{T} \to (c : T\ \ell\ viz\ M') :: \mathbb{T}} \qquad \text{(RED TABLE RIGHT)}\quad \frac{\mathbb{T} \to \mathbb{T}' \quad \mathsf{Core}(\mathsf{col})}{\mathsf{col} :: \mathbb{T} \to \mathsf{col} :: \mathbb{T}'}$$

$$\text{(RED INDEX INNER)}\quad \frac{M \to M'}{M[e_{index} < e_{size}] \to M'[e_{index} < e_{size}]} \qquad \text{(RED INDEX)}\quad \frac{\mathsf{Core}(\mathbb{T}) \quad \mathrm{fv}(e_{index}, e_{size}) \cap (\mathrm{dom}(\mathbb{T})) = \varnothing}{(\mathbb{T}\ R)[e_{index} < e_{size}] \to (\mathbb{T}[e_{index} < e_{size}]\ R)}$$

$$\text{(RED INDEX EXPR)}\quad \frac{}{E[e_{index} < e_{size}] \to E} \qquad \text{(RED APPL) (for Core}(\mathbb{T}))\quad \frac{((\mathbb{T} \wedge \ell) \wedge viz) \vdash R \rightsquigarrow_o \mathbb{T}_1 \qquad (\mathrm{locals}(\mathbb{T}) \cup \mathrm{inputs}(\mathbb{T})) \cap (\mathrm{fv}(\mathbb{T}') \cup \mathrm{dom}(\mathbb{T}')) = \varnothing}{(o : T'\ \ell\ viz\ (\mathbb{T}\ R)) :: \mathbb{T}' \to \mathbb{T}_1 @ \mathbb{T}'}$$

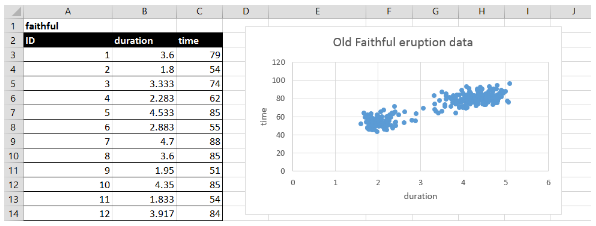
.

The purpose of the side-condition on (RED APPL) is to avoid variable-capture after reduction. The set $\mathrm{outputs}(\mathbb{T} \wedge viz)$ consists of the fields exported from the body of the function. The set $\mathrm{locals}(\mathbb{T}) \cap \mathrm{fv}(\mathbb{T}')$ consists of the variables in $\mathbb{T}'$ that are captured by inlining the binders in $\mathbb{T}$; there should be no more of these captures than expected by the exports of the function body.

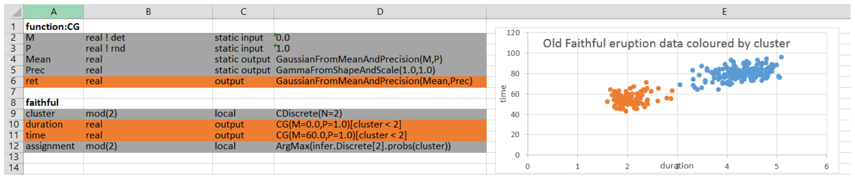
By using the above rules to expand out the three function calls and the two model expressions in the Old Faithful example, we obtain the core model below:

table faithful			
cluster_V	**real**[2]!**rnd**	**static output**	Dirichlet[2]([**for** i < 2 → 1.0])
cluster	**mod**(2)!**rnd**	**output**	Discrete[2](cluster_V)
duration_Mean	**real**[2]!**rnd**	**static output**	[**for** _ < 2 → GaussianFromMeanAndPrecision(0.0, 1.0)]
duration_Prec	**real**[2]!**rnd**	**static output**	[**for** _ < 2 → Gamma(1.0, 1.0)]
duration	**real**!**rnd**	**output**	GaussianFromMeanAndPrecision(duration_Mean[cluster], duration_Prec[cluster])
time_Mean	**real**[2]!**rnd**	**static output**	[**for** _ < 2 → GaussianFromMeanAndPrecision(60.0, 1.0)]
time_Prec	**real**[2]!**rnd**	**static output**	[**for** _ < 2 → Gamma(1.0, 1.0)]
time	**real**!**rnd**	**output**	GaussianFromMeanAndPrecision(time_Mean[cluster], time_Prec[cluster])
assignment	**mod**(2)!**qry**	**output**	ArgMax(**infer**.Discrete[2].probs(cluster))

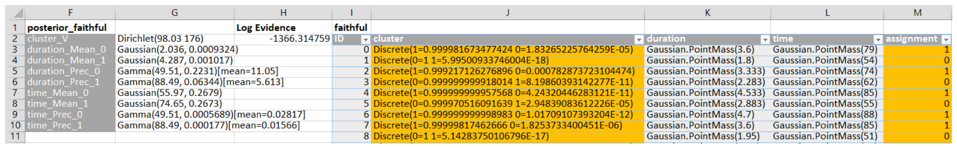
Moreover, here is a screen shot (best viewed in colour) of the model in Excel.



	A	B	C
1	**faithful**		
2	**ID**	**duration**	**time**
3	1	3.6	79
4	2	1.8	54
5	3	3.333	74
6	4	2.283	62
7	5	4.533	85
8	6	2.883	55
9	7	4.7	88
10	8	3.6	85
11	9	1.95	51
12	10	4.35	85
13	11	1.833	54
14	12	3.917	84



	A	B	C	D
1	**function:CG**			
2	M	real ! det	static input	0.0
3	P	real ! rnd	static input	1.0
4	Mean	real	static output	GaussianFromMeanAndPrecision(M,P)
5	Prec	real	static output	GammaFromShapeAndScale(1.0,1.0)
6	ret	real	output	GaussianFromMeanAndPrecision(Mean,Prec)
7				
8	**faithful**			
9	cluster	mod(2)	local	CDiscrete(N=2)
10	duration	real	output	CG(M=0.0,P=1.0)[cluster < 2]
11	time	real	output	CG(M=60.0,P=1.0)[cluster < 2]
12	assignment	mod(2)	local	ArgMax(infer.Discrete[2].probs(cluster))
13				
14				



	F	G	H	I	J	K	L	M
1	posterior_faithful		Log Evidence	faithful				
2	cluster_V	Dirichlet(98.03 176)	-1366.314759	ID	cluster	duration	time	assignment
3	duration_Mean_0	Gaussian(2.036, 0.0009324)		0	Discrete(1=0.999981673477424 0=1.83265225764259E-05)	Gaussian.PointMass(3.6)	Gaussian.PointMass(79)	1
4	duration_Mean_1	Gaussian(4.287, 0.001017)		1	Discrete(0=1 1=5.99500933746004E-18)	Gaussian.PointMass(1.8)	Gaussian.PointMass(54)	0
5	duration_Prec_0	Gamma(49.51, 0.2231)[mean=11.05]		2	Discrete(1=0.999217126276896 0=0.000782873723104474)	Gaussian.PointMass(3.333)	Gaussian.PointMass(74)	1
6	duration_Prec_1	Gamma(88.49, 0.06344)[mean=5.613]		3	Discrete(0=0.999999999918014 1=8.19860393142277E-11)	Gaussian.PointMass(2.283)	Gaussian.PointMass(62)	0
7	time_Mean_0	Gaussian(55.97, 0.2679)		4	Discrete(1=0.999999999957568 0=4.24320446283121E-11)	Gaussian.PointMass(4.533)	Gaussian.PointMass(85)	1
8	time_Mean_1	Gaussian(74.65, 0.2673)		5	Discrete(0=0.999970516091639 1=2.94839083612226E-05)	Gaussian.PointMass(2.883)	Gaussian.PointMass(55)	0
9	time_Prec_0	Gamma(49.51, 0.0005689)[mean=0.02817]		6	Discrete(1=0.999999999998983 0=1.01709107393204E-12)	Gaussian.PointMass(4.7)	Gaussian.PointMass(88)	1
10	time_Prec_1	Gamma(88.49, 0.000177)[mean=0.01566]		7	Discrete(1=0.99999817462666 0=1.8253733400451E-06)	Gaussian.PointMass(3.6)	Gaussian.PointMass(85)	1
11				8	Discrete(0=1 1=5.14283750106796E-17)	Gaussian.PointMass(1.95)	Gaussian.PointMass(51)	0

5 Dependent Type System and Semantics

5.1 Dependent Type System

The expressions of Tabular are based on the probabilistic language Fun [4]. We significantly extend Fun by augmenting its types with the three spaces described in Section 1, adding value-indexed dependent types including statically-bounded integers and sized arrays, and additional expressions including an operator for inference and operations for referencing attributes of tables and their instances.

We use *table types* Q both for functions and for concrete tables. When used to type a function Q must satisfy the predicate $\textsf{fun}(Q)$, which requires it to use the distinguished name ret for the explicit result of the function (its final output). When used to type a concrete table Q must satisfy the predicate $\textsf{table}(Q)$, which ensures that types do not depend on the contents of any input table t (except for the sizes of tables). We only need $\textsf{table}(Q)$ to define a conformance relation on databases and schema types. The schema type $Q_{CDiscrete}$ is listed in Section 1 in a grid form, instead of the textual notation below.

Table and Schema Types:

$Q ::= [(c_i : T_i\ \ell_i\ viz_i)\ ^{i \in 1..n}]$	table type (c_i distinct, $viz_i \neq \textsf{local}$)
$Sty ::= [(t_i : Q_i)\ ^{i \in 1..n}]$	schema type (t_i distinct)

$\textsf{fun}(Q)$ iff $viz_n = \textsf{output}$ and $c_n = \mathsf{ret}$.
$\textsf{model}(Q)$ iff $\textsf{fun}(Q)$ and each $viz_i = \textsf{output}$.
$\textsf{table}(Q)$ iff for each $i \in 1..n$, $\ell_i = \textsf{static} \Rightarrow \textsf{rnd}(T_i) \vee \textsf{qry}(T_i)$.

Free Variables: $\mathrm{fv}(Q)$

$\mathrm{fv}([]) = \varnothing$
$\mathrm{fv}(((c : T\ \ell\ viz)) :: Q) = \mathrm{fv}(T) \cup (\mathrm{fv}(Q) \setminus \{c\})$

Tabular typing environments Γ are ordered maps associating variables with their declared level and type, and table identifiers with their inferred table types. The typing rules will prevent expressions typed at level **static** from referencing **inst** level variables.

Tabular Typing Environments:

$\Gamma ::= \varnothing \mid (\Gamma, x :^{\ell} T) \mid (\Gamma, t : Q)$	environment
$\gamma([(c_i : T_i\ \ell_i\ viz_i)\ ^{i \in 1..n}]) \triangleq c_i :^{\ell_i} T_i\ ^{i \in 1..n}$	Q as an environment

Next, we present the judgments and rules of the type system.

Judgments of the Tabular Type System:

$\Gamma \vdash \diamond$	environment Γ is well-formed
$\Gamma \vdash T$	in Γ, type T is well-formed
$\Gamma \vdash^{pc} e : T$	in Γ at level pc, index expression e has type T
$\Gamma \vdash Q$	in Γ, table type Q is well-formed
$\Gamma \vdash Sty$	in Γ, schema type Sty is well-formed
$\Gamma \vdash T <: U$	in Γ, T is a subtype of U
$\Gamma \vdash^{pc} E : T$	in Γ at level pc, expression E has type T
$\Gamma \vdash^{pc}_{o} R : Q \to Q'$	R sends function type Q to model type Q' in column o
$\Gamma \vdash^{pc}_{o} M : Q$	model expression M has model type Q
$\Gamma \vdash^{pc} \mathbb{T} : Q$	table $\mathbb{T}$ has type Q
$\Gamma \vdash \mathbb{S} : Sty$	schema $\mathbb{S}$ has type Sty

The formation rules for types and environments depend mutually on the typing rules for index expressions. Only index expressions that are **det**-space and **static**-level may occur in types. We write $\mathsf{ty}(s)$ for the scalar type S of the scalar s.

Rules for Types, Environments, and Index Expressions: $\Gamma \vdash \diamond \quad \Gamma \vdash T \quad \Gamma \vdash^{pc} e : T$

(Env Empty)
$$\frac{}{\varnothing \vdash \diamond}$$

(Env Var)
$$\frac{\Gamma \vdash T \quad x \notin \mathrm{dom}(\Gamma)}{\Gamma, x :^{pc} T \vdash \diamond}$$

(Env Table) ($\mathbf{table}(Q)$)
$$\frac{\Gamma \vdash Q \quad t \notin \mathrm{dom}(\Gamma)}{\Gamma, t : Q \vdash \diamond}$$

(Type Scalar)
$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash S \,!\, spc}$$

(Type Range)
$$\frac{\Gamma \vdash^{\mathbf{static}} e : \mathbf{int} \,!\, \mathbf{det}}{\Gamma \vdash \mathbf{mod}(e) \,!\, spc}$$

(Type Array)
$$\frac{\Gamma \vdash T \quad \Gamma \vdash^{\mathbf{static}} e : \mathbf{int} \,!\, \mathbf{det}}{\Gamma \vdash T[e]}$$

(Index Var) (for $\ell \leq pc$)
$$\frac{\Gamma \vdash \diamond \quad \Gamma = \Gamma_1, x :^{\ell} T, \Gamma_2}{\Gamma \vdash^{pc} x : T}$$

(Index Scalar)
$$\frac{\Gamma \vdash \diamond \quad S = \mathsf{ty}(s)}{\Gamma \vdash^{pc} s : S \,!\, \mathbf{det}}$$

(Index Mod)
$$\frac{\Gamma \vdash \diamond \quad 0 \leq n < m}{\Gamma \vdash^{pc} n : \mathbf{mod}(m) \,!\, \mathbf{det}}$$

(Index Sizeof)
$$\frac{\Gamma \vdash \diamond \quad \Gamma = \Gamma', t : Q, \Gamma''}{\Gamma \vdash^{pc} \mathbf{sizeof}(t) : \mathbf{int} \,!\, \mathbf{det}}$$

Formation Rules for Table and Schema Types: $\Gamma \vdash Q \quad \Gamma \vdash Sty$

(Table Type [])
$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash [] : []}$$

(Table Type Input)
$$\frac{\Gamma \vdash T \quad \Gamma, c :^{\ell} T \vdash Q}{\Gamma \vdash (c : T\ \ell\ \mathbf{input}) :: Q}$$

(Table Type Output)
$$\frac{\Gamma \vdash T \quad \Gamma, c :^{\ell} T \vdash Q}{\Gamma \vdash (c : T\ \ell\ \mathbf{output}) :: Q}$$

(Schema Type [])
$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash [] : []}$$

(Schema Type Table)
$$\frac{\Gamma \vdash Q \quad \mathbf{table}(Q) \quad \Gamma, t : Q \vdash Sty}{\Gamma \vdash (t : Q) :: Sty}$$

Subtyping allows **det**-space data to be used as **rnd**-space or **qry**-space data. The preorder $\leq$ on spaces is the least reflexive relation to satisfy $\mathbf{det} \leq \mathbf{rnd}$ and $\mathbf{det} \leq \mathbf{qry}$. The default space is **det**, so when we write S or $\mathbf{mod}(e)$ as a type, we mean $S \,!\, \mathbf{det}$ or $\mathbf{mod}(e) \,!\, \mathbf{det}$. We define a commutative partial operation $spc \vee spc'$, and lift this operation to types $T \vee spc$ to weaken the space of a type.

Least upper bound: $spc \vee spc'$ **(if** $spc \leq spc'$ **or** $spc' \leq spc$**)**

$spc \vee spc = spc$ $\quad$ $\mathsf{det} \vee \mathsf{rnd} = \mathsf{rnd}$ $\quad$ $\mathsf{det} \vee \mathsf{qry} = \mathsf{qry}$
(The combination $\mathsf{rnd} \vee \mathsf{qry}$ is intentionally not defined.)

Operations on Types and Spaces: $T \vee spc$

$(S \,!\, spc) \vee spc' \triangleq S \,!\, (spc \vee spc')$ $\quad$ $T[e] \vee spc \triangleq (T \vee spc)[e]$
$(\mathsf{mod}(e) \,!\, spc) \vee spc' \triangleq \mathsf{mod}(e) \,!\, (spc \vee spc')$

Given these definitions, we present the rules of subtyping and of typing expressions.

Rules of Subtyping: $\Gamma \vdash T <: U$

(SUB SCALAR)
$$\frac{\Gamma \vdash \diamond \quad spc_1 \leq spc_2}{\Gamma \vdash S \,!\, spc_1 <: S \,!\, spc_2}$$

(SUB MOD)
$$\frac{\Gamma \vdash^{\mathsf{static}} e : \mathsf{int} \,!\, \mathsf{det} \quad spc_1 \leq spc_2}{\Gamma \vdash \mathsf{mod}(e) \,!\, spc_1 <: \mathsf{mod}(e) \,!\, spc_2}$$

(SUB ARRAY)
$$\frac{\Gamma \vdash T <: U \qquad \Gamma \vdash^{\mathsf{static}} e : \mathsf{int} \,!\, \mathsf{det}}{\Gamma \vdash T[e] <: U[e]}$$

As mentioned in Section 3, we assume a collection of total deterministic functions g, including arithmetic and logical operators. Below, the scalar types all have the default space of **det**. We have equality on integers, but not on reals. For each function symbol g, we write $\underline{g}$ for its meaning as a total function on scalars.

Deterministic Primitives: $g_{spc} : (x_1 : T_1, \ldots, x_n : T_n) \to T$

$(>)_{spc} : (\mathsf{x1} : \mathsf{real}!spc, \mathsf{x2} : \mathsf{real}!spc) \to \mathsf{bool}!spc$
$(>_{spc} : (\mathsf{x1} : \mathsf{int}!spc, \mathsf{x2} : \mathsf{int}!spc) \to \mathsf{bool}!spc$
$(=)_{spc} : (\mathsf{x1} : \mathsf{int}!spc, \mathsf{x2} : \mathsf{int}!spc,) \to \mathsf{bool}!spc$
$\mathsf{or}_{spc} : (\mathsf{x1} : \mathsf{bool}!spc, \mathsf{x2} : \mathsf{bool}!spc,) \to \mathsf{bool}!spc$
$(-)_{spc} : (\mathsf{x1} : \mathsf{real}!spc, \mathsf{x2} : \mathsf{real}!spc) \to \mathsf{real}!spc$
$(-)_{spc} : (\mathsf{x1} : \mathsf{int}!spc, \mathsf{x2} : \mathsf{int}!spc) \to \mathsf{int}!spc$

Here is a set of primitive distributions with their dependent signatures.

Distributions: $D_{spc} : [x_1 : T_1, \ldots, x_m : T_m](y_1 : U_1, \ldots, y_n : U_n) \to T$

$\mathsf{Bernoulli}_{spc} : (\mathsf{bias} : \mathsf{real}!spc) \to \mathsf{bool}!\mathsf{rnd}$
$\mathsf{Beta}_{spc} :: (\mathsf{a} : \mathsf{real}!spc, \mathsf{b} : \mathsf{real}!spc) \to \mathsf{real}!\mathsf{rnd}$
$\mathsf{Discrete}_{spc} : [\mathsf{N} : \mathsf{int}!\mathsf{det}](\mathsf{probs} : \mathsf{real}!spc[\mathsf{N}]) \to \mathsf{mod}(\mathsf{N})!\mathsf{rnd}$
$\mathsf{Dirichlet}_{spc} : [\mathsf{N} : \mathsf{int}!\mathsf{det}](\mathsf{pseudocount} : (\mathsf{real}!spc)[\mathsf{N}]) \to (\mathsf{real}!\mathsf{rnd})[\mathsf{N}]$
$\mathsf{Gamma}_{spc} : (\mathsf{shape} : \mathsf{real}!spc, \mathsf{scale} : \mathsf{real}!spc) \to \mathsf{real}!\mathsf{rnd}$
$\mathsf{Gaussian}_{spc} : (\mathsf{mean} : \mathsf{real}!spc, \mathsf{variance} : \mathsf{real}!spc) \to \mathsf{real}!\mathsf{rnd}$
$\mathsf{VectorGaussian}_{spc} :$
$[\mathsf{N} : \mathsf{int}!\mathsf{det}](\mathsf{mean} : (\mathsf{real}!spc)[\mathsf{N}], \mathsf{covariance} : \mathsf{real}!spc[\mathsf{N}][\mathsf{N}]) \to$
$\quad (\mathsf{real}!\mathsf{rnd}[\mathsf{N}])$

Lemma 1. *Whenever $D_{spc} : [x_1 : T_1, \dots, x_m : T_m](y_1 : U_1, \dots, y_n : U_n) \to T$, each T_i is in* $\textsf{det}$*-space, each U_i is in spc-space, and T is in* $\textsf{rnd}$*-space.*

Proof: By inspection. ■

Typing Rules for Expressions: $\Gamma \vdash^{pc} E : T$

(SUBSUM)
$$\frac{\Gamma \vdash^{pc} E : T \quad \Gamma \vdash T <: U}{\Gamma \vdash^{pc} E : U}$$

(INDEX EXPRESSION)
$$\frac{\Gamma \vdash^{pc} e : T \quad E \equiv e}{\Gamma \vdash^{pc} E : T}$$

(DEREF STATIC)
$$\frac{\Gamma = \Gamma', t : Q, \Gamma'' \qquad Q = Q' @ [(c : T\ \textsf{static}\ viz)] @ Q''}{\Gamma \vdash^{pc} t.c : T}$$

(DEREF INST)
$$\frac{\Gamma \vdash^{pc} E : \textsf{link}(t)\,!\,spc \qquad \Gamma = \Gamma', t : Q, \Gamma'' \quad Q = Q' @ [(c : T\ \textsf{inst}\ viz)] @ Q''}{\Gamma \vdash^{pc} E : t.c : T \vee spc}$$

(RANDOM) (where $\sigma(U) \triangleq U\{^{e_1}/_{x_1}\}\dots\{^{e_m}/_{x_m}\}$)
$$\frac{\begin{array}{l} D_{\textsf{rnd}} : [x_1 : T_1, \dots, x_m : T_m](y_1 : U_1, \dots, y_n : U_n) \to T \\ \Gamma \vdash^{\textsf{static}} e_i : T_i \quad \forall i \in 1..m \quad \Gamma \vdash^{pc} F_j : \sigma(U_j) \quad \forall j \in 1..n \quad \Gamma \vdash \diamond \\ \{x_1, \dots, x_m\} \cap (\bigcup_i \text{fv}(e_i)) = \varnothing \quad x_i \neq x_j \text{ for } i \neq j \end{array}}{\Gamma \vdash^{pc} D[e_1, \dots, e_m](F_1, \dots, F_n) : \sigma(T)}$$

(ITER) (where $x \notin \text{fv}(T)$)
$$\frac{\Gamma \vdash^{\textsf{static}} e : \textsf{int}\,!\,\textsf{det} \qquad \Gamma, x :^{pc} (\textsf{mod}(e)\,!\,\textsf{det}) \vdash^{pc} F : T}{\Gamma \vdash^{pc} [\textsf{for}\ x < e \to F] : T[e]}$$

(INDEX)
$$\frac{\text{space}(T) \leq spc \qquad \Gamma \vdash^{pc} E : T[e] \quad \Gamma \vdash^{pc} F : \textsf{mod}(e)\,!\,spc}{\Gamma \vdash^{pc} E[F] : T \vee spc}$$

(INFER) (where $\sigma(U) \triangleq U\{^{e_1}/_{x_1}\}\dots\{^{e_m}/_{x_m}\}$)
$$\frac{\begin{array}{l} D_{\textsf{qry}} : [x_1 : T_1, \dots, x_m : T_m](y_1 : U_1, \dots, y_n : U_n) \to T \\ \Gamma \vdash^{\textsf{static}} e_i : T_i \quad \forall i \in 1..m \quad \Gamma \vdash^{pc} x : \sigma(T) \quad j \in 1..n \\ \{x_1, \dots, x_m\} \cap (\bigcup_i \text{fv}(e_i)) = \varnothing \quad x_i \neq x_j \text{ for } i \neq j \end{array}}{\Gamma \vdash^{pc} \textsf{infer}.D[e_1, \dots, e_m].y_j(x) : \sigma(U_j)}$$

(PRIM)
$$\frac{\begin{array}{l} \Gamma \vdash \diamond \quad \sigma_i(T) \triangleq T\{^{E_j}/_{x_j}\}^{j \in 1..i-1} \quad spc \in \{\textsf{det}, \textsf{rnd}, \textsf{qry}\} \\ g : (x_1 : T_1, \dots, x_n : T_n) \to T \quad \Gamma \vdash^{pc} E_i : \sigma_i(T_i \vee spc) \quad \forall i \in 1..n \\ \{x_1, \dots, x_n\} \cap (\bigcup_i \text{fv}(E_i)) = \varnothing \quad x_i \neq x_j \text{ for } i \neq j \end{array}}{\Gamma \vdash^{pc} g(E_1, \dots, E_n) : \sigma_{n+1}(T \vee spc)}$$

(IF)
$$\frac{\Gamma \vdash^{pc} E_1 : (\textsf{bool}\,!\,spc) \quad \Gamma \vdash^{pc} E_2 : T \quad \Gamma \vdash^{pc} E_3 : T \quad \text{space}(T) \leq spc}{\Gamma \vdash^{pc} \textsf{if}\ E_1\ \textsf{then}\ E_2\ \textsf{else}\ E_3 : T \vee spc}$$

(ARRAY)
$$\frac{\Gamma \vdash \diamond \quad \Gamma \vdash^{pc} E_i : T \quad \forall i \in 0..n-1}{\Gamma \vdash^{pc} [E_0, \dots, E_{n-1}] : T[n]}$$

For an example of (INFER), recall the expression **infer**.Dirichlet[2].counts(V) from Section 1. Here $m = n = 1$, $y_1 =$ counts and $U_1 = \textbf{real}[\mathsf{N}]!\textbf{qry}$ and $\sigma = \{{}^{2}/_{\mathsf{N}}\}$ and the result type is $\sigma(U_1) = \textbf{real}[2]!\textbf{qry}$. In (PRIM), notice that all the types T_i, T occuring in the signature of the deterministic operator g are in **det**-space. Hence, the types $T_i \vee spc$ and $T \vee spc$ are well-defined.

Typing Rules for Arguments: $\Gamma \vdash_o^{pc} R : Q \to Q'$

(ARG INPUT)
$$\frac{\Gamma \vdash^{\ell \wedge pc} e : T \quad \Gamma \vdash_o^{pc} R : Q\{{}^{e}/_{c}\} \to Q'}{\Gamma \vdash_o^{pc} ((c = e) :: R) : ((c : T\ \ell\ \textbf{input}) :: Q) \to Q'}$$

(ARG OUTPUT)
$$\frac{\Gamma \vdash T \qquad \Gamma \vdash_o^{pc} R : Q\{{}^{o\_c}/_{c}\} \to Q' \qquad c \neq \mathsf{ret}}{\Gamma \vdash_o^{pc} R : ((c : T\ \ell\ \textbf{output}) :: Q) \to ((o\_c : T\ (\ell \wedge pc)\ \textbf{output}) :: Q')}$$

(ARG RET)
$$\frac{\Gamma \vdash T}{\Gamma \vdash_o^{pc} R : (\mathsf{ret} : T\ \ell\ \textbf{output}) \to (\mathsf{ret} : T\ (\ell \wedge pc)\ \textbf{output})}$$

For example, if $Q_{CDiscrete}$ is the function type of CDiscrete from Section 1 we can derive $\mathsf{b} :^{\textbf{static}} \textbf{real}!\textbf{rnd} \vdash^{\textbf{inst}}_{\mathsf{Flip}} (\mathsf{N} = 2, \mathsf{alpha} = \mathsf{b}) : Q_{CDiscrete} \to Q'$ where Q', shown in the grid below, represents the outputs of the function call. Since the inputs N and alpha of CDiscrete are both **static**, arguments 2 and b are typed at level **static** $\wedge$ **inst** $=$ **static**.

Flip_V	**real**[2]!**rnd**	**static output**
ret	**mod**(2)!**rnd**	**output**

Next, we have rules for assigning a model type Q to a model expression M. (MODEL INDEXED) needs the following operation $Q[e]$ to capture the static effect of indexing: It allows e_{index} to be in any space spc, but the typechecker might show a warning if $spc \neq \textbf{rnd}$.

Indexing a Table Type: $Q[e]$

$$\varnothing[e] \triangleq \varnothing$$
$$((c : T\ \textbf{inst}\ viz) :: Q)[e] \triangleq (c : T\ \textbf{inst}\ viz) :: (Q[e])$$
$$((c : T\ \textbf{static}\ viz) :: Q)[e] \triangleq (c : T\ \textbf{static}\ viz) :: (Q[e]) \quad \text{if } viz = \textbf{input} \text{ or } \neg\textbf{rnd}(T)$$
$$((c : T\ \textbf{static}\ viz) :: Q)[e] \triangleq (c : T[e]\ \textbf{static}\ viz) :: (Q[e]) \quad \text{if } viz \neq \textbf{input} \text{ and } \textbf{rnd}(T)$$

The vectorized c cannot appear in Q when $\textbf{rnd}(T)$, so $Q[e]$ remains well-formed.

Typing Rules for Model Expressions: $\Gamma \vdash_o^{pc} M : Q$

(MODEL EXPRESSION)
$$\frac{\Gamma \vdash^{pc} E : T}{\Gamma \vdash_o^{pc} E : [(\mathsf{ret} : T\ pc\ \textbf{output})]}$$

(MODEL APPL)
$$\frac{\Gamma \vdash^{pc} \mathbb{T} : Q \quad \textbf{fun}(Q) \quad \Gamma \vdash_o^{pc} R : Q \to Q'}{\Gamma \vdash_o^{pc} \mathbb{T}\ R : Q'}$$

(MODEL INDEXED)

$$\frac{\Gamma \vdash^{pc}_{o} M : Q \quad \mathrm{dom}(Q) \cap \mathrm{fv}(e_{size}) = \varnothing \quad \Gamma \vdash^{pc} e_{index} : \mathbf{mod}(e_{size})\,!\,spc}{\Gamma \vdash^{pc}_{o} M[e_{index} < e_{size}] : Q[e_{size}]}$$

Finally, we complete the system with rules for tables and schemas.

Typing Rules for Tables: $\Gamma \vdash^{pc} \mathbb{T} : Q$

(TABLE []) (TABLE INPUT)

$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash^{pc} [] : []} \qquad \frac{\Gamma, c :^{\ell \wedge pc} T \vdash^{pc} \mathbb{T} : Q}{\Gamma \vdash^{pc} (c : T\ \ell\ \mathbf{input}\ \varepsilon) :: \mathbb{T} : (c : T\ (\ell \wedge pc)\ \mathbf{input}) :: Q}$$

(TABLE OUTPUT)

$$\frac{\Gamma \vdash^{\ell \wedge pc}_{c} M : Q_c @[(\mathsf{ret} : T\ (\ell \wedge pc)\ \mathbf{output})] \quad \Gamma, \gamma(Q_c), c :^{\ell \wedge pc} T \vdash^{pc} \mathbb{T} : Q}{\Gamma \vdash^{pc} (c : T\ \ell\ \mathbf{output}\ M) :: \mathbb{T} : Q_c @((c : T\ (\ell \wedge pc)\ \mathbf{output}) :: Q)}$$

(TABLE LOCAL) (where $(\mathrm{dom}(Q_c) \cup \{c\}) \cap \mathrm{fv}(Q) = \varnothing$)

$$\frac{\Gamma \vdash^{\ell \wedge pc}_{c} M : Q_c @[(\mathsf{ret} : T\ (\ell \wedge pc)\ \mathbf{output})] \quad \Gamma, \gamma(Q_c), c :^{\ell \wedge pc} T \vdash^{pc} \mathbb{T} : Q}{\Gamma \vdash^{pc} (c : T\ \ell\ \mathbf{local}\ M) :: \mathbb{T} : Q}$$

Typing Rules for Schemas: $\Gamma \vdash \mathbb{S} : Sty$

(SCHEMA []) (SCHEMA TABLE)

$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash [] : []} \qquad \frac{\Gamma \vdash^{\mathsf{inst}} \mathbb{T} : Q \quad \mathbf{table}(Q) \quad \Gamma, t : Q \vdash \mathbb{S} : Sty}{\Gamma \vdash (t = \mathbb{T}) :: \mathbb{S} : (t : Q) :: Sty}$$

5.2 Reduction to Core Tabular

Proposition 1 (Preservation).

(1) *If $\Gamma \vdash \mathbb{S} : Sty$ and $\mathbb{S} \to \mathbb{S}'$ then $\Gamma \vdash \mathbb{S}' : Sty$.*
(2) *If $\Gamma \vdash^{pc} \mathbb{T} : Q$ and $\mathbb{T} \to \mathbb{T}'$ then $\Gamma \vdash^{pc} \mathbb{T}' : Q$.*
(3) *If $\Gamma \vdash^{pc} M : Q$ and $M \to M'$ then $\Gamma \vdash^{pc} M' : Q$.*

Proof: See Appendix F.2. ■

Proposition 2 (Progress). *If $\Gamma \vdash^{pc} \mathbb{S} : Q$ either* Core($\mathbb{S}$) *or there is $\mathbb{S}'$ such that $\mathbb{S} \to \mathbb{S}'$.*

Proof: See Appendix F.3. ■

Proposition 3 (Termination). *No infinite chain $\mathbb{S}_0 \to \mathbb{S}_1 \to \ldots$ exists.*

Proof: See Appendix F.4 ■

Algorithm 1: Reducing to Core Schema: $\mathsf{CoreSchema}(\mathbb{S})$

```
(1) Compute $\mathbb{S}'$ such that $\mathbb{S} \to^* \mathbb{S}'$ and $\mathsf{Core}(\mathbb{S}')$.
(2) Output $\mathbb{S}'$.
```

As a corollary of our three propositions, we obtain:

Theorem 1. *If* $\varnothing \vdash \mathbb{S} : Sty$ *then* $\mathsf{CoreSchema}(\mathbb{S})$ *terminates with a unique schema* $\mathbb{S}'$ *such that* $\mathbb{S} \to^* \mathbb{S}'$ *and* $\mathrm{Core}(\mathbb{S}')$ *and* $\varnothing \vdash \mathbb{S}' : Sty$.

Proof: By Proposition 2, if $\varnothing \vdash \mathbb{S} : Sty$ and there is no $\mathbb{S}'$ such that $\mathbb{S} \to \mathbb{S}'$, then $\mathrm{Core}(\mathbb{S})$. By Proposition 3, there is no infinite chain $\mathbb{S} \to \mathbb{S}_1 \to \cdots \to \mathbb{S}_i \to \ldots$. The uniqueness of $\mathbb{S}'$ follows from the fact that the reduction rules are deterministic. Meanwhile, $\varnothing \vdash \mathbb{S}' : Sty$ follows from Proposition 1. ■

Although schemas define probabilistic models, reduction is deterministic because it simply unravels model expressions into their core form, without any probabilistic computation.

5.3 Semantics of Core Tabular (Sketch)

Following [4], we define a semantics based on measure theory for **det** and **rnd**-level attributes, plus a set of evaluation rules for **qry**-level variables. For the sake of brevity, we omit the precise definitions here, and instead sketch the semantics and state the key theoretical result, illustrating it by example. We list the full details in Appendix E.

The denotational semantics of a schema $\mathbb{S}$ with respect to an input database δ_{in} is a measure μ defined on the measurable space corresponding to this schema. In order to evaluate the queries in the schema, we need to compute marginal measures for all (non-**qry**) attributes of all tables. We do so dynamically by the query evaluation rules, which construct for each table an environment ρ storing the marginal measures for all random attributes, as well as values for non-random ones.

More precisely, our semantics for Tabular factors into an idealised, probabilistic denotational semantics (abstracting the details of approximate inference algorithms such as Infer.NET and other potential implementations) and a mostly conventional operational semantics.

The denotational semantics interprets well-typed schema as inductively defined measurable spaces, $\boldsymbol{T}[\![\mathbb{S}]\!]^{\rho_{sz}}$, and a (mathematical) function interpreting well typed schemas $\boldsymbol{P}[\![\mathbb{S}]\!]^{\delta_{in}}_{(\tau,\delta)} \in \boldsymbol{T}[\![\mathbb{S}]\!]^{\rho_{sz}}$ as sub-probability measures describing the joint distribution μ of random variables given the observed input database δ_{in}.

The relation $\delta \vdash_{\sigma} \mathbb{S} \Downarrow \delta_{out}$ of our operational semantics takes as input marginal measures μ_i in σ for the tables and database (respectively), and the current operational environment δ (mapping **qry** and **det** attributes to values and **rnd** attributes to marginals), and a schema. It returns an (output) database value δ_{out}: a nested map that assigns values to each non-**rnd** attribute of the schema.

Algorithm 2: Query Semantics of Core Schema: $\mathsf{CoreQuery}(\mathbb{S}, DB)$

```
(1) Assume $\mathsf{core}(\mathbb{S})$, $\mathbb{S} = [(t_1 = \mathbb{T}_1), \ldots, (t_n = \mathbb{T}_n)]$, and $DB = (\delta_{in}, \rho_{sz})$.
(2) Let $\mu \triangleq \boldsymbol{P}[\![\mathbb{S}]\!]_{[]}^{\delta_{in}}$ (that is, the joint distribution over all rnd-variables).
(3) Let $\mu_i \triangleq \mathsf{marg}(\mu, i, n)$ (that is, the marginal distribution for each table $t_i$).
(4) Let $\hat{\sigma} \triangleq [t_i \to \mu_i^{i \in 0..(n-1)}]$.
(5) Return $(\delta_{out}, \rho_{sz})$ such that $\varnothing \vdash_{\hat{\sigma}} \mathbb{S} \Downarrow \delta_{out}$.
```

Theorem 2 below states that, given a well-typed schema and conforming database, the composition of the denotational semantics and the deterministic evaluation relation yields a well-typed output database (with the same dimensions). The notation $DB \models^{\mathsf{in}} Sty$ means that the database DB is a well-formed input to Sty; dually, $DB \models^{\mathsf{out}} Sty$ means that the database DB is a well-formed output of Sty.

Theorem 2. *Suppose* $\mathrm{Core}(\mathbb{S})$ *and* $\varnothing \vdash \mathbb{S} : Sty$ *and* $DB = (\delta_{in}, \rho_{sz})$ *and* $DB \models^{\mathsf{in}} Sty$. *Then algorithm* $\mathsf{CoreQuery}(\mathbb{S}, DB)$ *returns* $DB' = (\delta_{out}, \rho_{sz})$ *such that* $DB' \models^{\mathsf{out}} Sty$.

To illustrate, consider the Old Faithful schema shown in Section 4, together with an input database $(\delta_{in}, [\mathsf{faithful} \mapsto 272])$ with 272 rows (say) such as the following:

$$\delta_{in} = [\,\mathsf{faithful} \mapsto [\,\mathsf{duration} \mapsto \mathbf{inst}([1.9;4.0;4.9;\ldots)];\;\; \mathsf{time} \mapsto \mathbf{inst}([50;75;80;\ldots])\,]\,]$$

The final environment constructed by evaluation of the only table in this schema is:

$$\begin{array}{l}
[\;\mathsf{cluster\_V} \mapsto \mathbf{static}(\mu_{\mathrm{Dirichlet}[2](1,1)});\\
\;\;\mathsf{cluster} \mapsto \mathbf{inst}([\mu_{20};\mu_{21};\mu_{22};\ldots]);\\
\;\;\mathsf{duration\_Mean} \mapsto \mathbf{static}([\mu_{\mathrm{Gaussian}(0,1)}, \mu_{\mathrm{Gaussian}(0,1)}]);\\
\;\;\mathsf{duration\_Prec} \mapsto \mathbf{static}([\mu_{\mathrm{Gamma}(1,1)}, \mu_{\mathrm{Gamma}(1,1)}]);\\
\;\;\mathsf{duration} \mapsto \mathbf{inst}([\mu_{50};\mu_{51};\mu_{52};\ldots]);\\
\;\;\mathsf{time\_Mean} \mapsto \mathbf{static}([\mu_{\mathrm{Gaussian}(60,1)}, \mu_{\mathrm{Gaussian}(60,1)}]);\\
\;\;\mathsf{time\_Prec} \mapsto \mathbf{static}([\mu_{\mathrm{Gamma}(1,1)}, \mu_{\mathrm{Gamma}(1,1)}]);\\
\;\;\mathsf{time} \mapsto \mathbf{inst}([\mu_{80};\mu_{81};\mu_{82};\ldots]);\\
\;\;\mathsf{assignment} \mapsto \mathbf{inst}([0;1;1;...])\;]
\end{array}$$

The output database is $(\delta_{out}, [\mathsf{faithful} \mapsto 272])$ where δ_{out} contains those entries from such environments which correspond to non-random attributes (that is, are not measures). In our example, it is of the form:

$$\delta_{out} = [\,\mathsf{faithful} \mapsto [\,\mathsf{assignment} \mapsto \mathbf{inst}([0;1;1;\ldots])\,]\,]$$

In this example, all of the **inst** arrays are of length 272.

6 Examples of Bayesian Decision Analysis in Tabular

To illustrate the value of query-space computations, we illustrate how they express a range of decision problems. Decisions such as these cannot be expressed in the original form of Tabular. Other probabilistic programming languages have built in constructs for decision-making, whereas Tabular does so using ideas of information flow.

We describe how three example decision problems are written as Tabular queries. The result of Bayesian inference is the posterior belief over quantities of interest, including model parameters such as the **rnd**-space variable V in our coins example. These inferences reflect a change of belief in light of data, but they are not sufficient for making decisions, which requires optimization under uncertainty.

In Bayesian Decision Analysis, the decision making process is based on statistical inference followed by maximization of expected utility of the outcome. Following Gelman et al. [10], Bayesian Decision Analysis can be described as follows:

(1) Enumerate sets D and X of all possible decision options $d \in D$ and outcomes $x \in X$.
(2) Determine the probability distribution over outcomes $x \in X$ conditional on each decision option $d \in D$.
(3) Define a utility function $U : X \to \mathbb{R}$ to value each outcome.
(4) Calculate the expected utility $E[U(x)|d]$ as a function of decision option d and make the decision with the highest expected utility.

In Tabular, the both the evaluation of the expected utility and the maximization can be expressed as query expressions, thus making it possible to perform the entire process of Bayesian Decision Analysis within the language.

(1) Optimal Betting Decisions Consider a situation in which to decide whether or not to accept a given sports bet based on the TrueSkill model for skill estimation [18]. The following code shows the schema $\mathbb{S}_{TrueSkill}$. Following [16], the tables Players and Matches generate **rnd**-space variables for the results of matches between players, by comparison of their per-match performances, modelled as noisy per-player skills.

table Players			
Skill	**real**!**rnd**	**output**	Gaussian(25.0,100.0)
table Matches			
Player1	**link**(Players)!**det**	**input**	
Player2	**link**(Players)!**det**	**input**	
Perf1	**real**!**rnd**	**output**	Gaussian(Player1.Skill,100.0)
Perf2	**real**!**rnd**	**output**	Gaussian(Player2.Skill,100.0)
Win1	**bool**!**rnd**	**output**	Perf1 > Perf2
table Bets			
Match	**link**(Matches)!**det**	**input**	
Odds1	**real**!**det**	**input**	
Win1	**bool**!**rnd**	**output**	Match.Win1
p	**real**!**qry**	**output**	**infer**.Bernoulli[].Bias(Win1)
U	**real**[3]!**det**	**output**	[0.0;−1.0;Odds1 ∗ 1.0]
EU	**real**[2]!**qry**	**output**	[U[0];((1.0 − p)∗ U[1])+ (p ∗ U[2])]
PlaceBet1	**mod**(2)!**qry**	**output**	ArgMax(EU)

Table Bets represents the decision theoretic part of the code and refers to Matches together with the odds Odds1 offered for a bet on player 1 winning. Here, the two decision options in $D = \{0,1\}$ are to take the bet (PlaceBet1 $= 1$) or not (PlaceBet1 $= 0$), and the three possible outcomes in $X = \{0,1,2\}$ are abstain $= 0$, loss $= 1$ or win $= 2$. The optimal decision depends on the odds: a risky bet may be worth taking if the odds are good. The utility function U is given by money won for a fixed bet size of, say, \$1, so $U(\text{abstain}) = 0.0$, $U(\text{loss}) = -\$1.0$, and $U(\text{win}) = \text{Odds1} * \$1.0$. Variable p is obtained from **qry** expression **infer**.Bernoulli.Bias(Win1) and represents the inferred probability of a positive bet outcome. The **qry** variable PlaceBet1 is 1 if the expected utility $EU[1] = (1-p)\cdot(-\$1.0) + p\cdot \text{Odds1}\cdot \$1.0$ is greater than $EU[0] = 0.0$, that is, betting is better than not betting. The ArgMax operator simply returns the first index (of type **mod**(n)) of the maximum value in its array argument (of type **real**[n]). It returns the decision delivering the maximum expected utility.

(2) Classes with Asymmetric Misclassification Costs Consider the task of n-ary classification with class-specific misclassification costs. We proceed by defining the schema for a Naive Bayes classifier (see, for instance, Duda and Hart [9]), in terms of the function CG (from Section refsec:reduction) which represents a Gaussian distribution, with **static** parameters assuming natural conjugate prior distributions. (A prior is called conjugate with respect to a likelihood if it takes the same functional form.)

Hence, we can write down the Naive Bayes model (for a simple gender classification task) very succinctly as follows (using the indexed model notation from Section 3).

table People			
g	**mod**(2)!**rnd**	**output**	CDiscrete(N=2,R=1.0)
height	**real**!**rnd**	**output**	(CG(M=0.0,P=1.0))[g<2]
weight	**real**!**rnd**	**output**	(CG(M=0.0,P=1.0))[g<2]
footsize	**real**!**rnd**	**output**	(CG(M=0.0,P=1.0))[g<2]
Us	**real**[2][2]!**qry**	**static output**	[[0.0;−20.0];[−10.0;0.0]]
action	**mod**(2)!**qry**	**output**	Action(N=2,UPT=Us,**class**=g)

The first four lines define a Naive Bayes model with Gaussian features height, weight, and footsize, which are assumed to be distributed as independent Gaussians conditional on knowing gender g. At this point, we could simply return the probability vector **infer**.Discrete[2].probs(g): the probabilities that a person has either gender.

However, suppose we need to return a concrete gender decision and that for some reason the cost of false positives differs from the cost of false negatives. Below we encode how to decide whether to take the action of predicting the gender of 0 (female) or 1 (male), given that: A false-positive (predict 1 but actually 0) costs 20. A false-negative (predict 0 but actually 1) costs 10. A true-positive or true-negative costs 0. The costs, expressed as negative utilities, are in the matrix Us.

The query defined by the model computes an action column, classifying each row, taking into account the relative costs of false positives and false negatives. (It recommends an action for all rows, even those already labelled with a gender.)

fun Action			
N	**int**!**det**	**static input**	
UPT	**real**[N][N]!**qry**	**static input**	
class	**mod**(N)!**rnd**	**input**	
probs	**real**[N]!**qry**	**output**	**infer**.Discrete[N].probs(**class**)
EU	**real**[N]!**qry**	**output**	[**for** p < N →Sum([**for** t < N →(probs[t] ∗ UPT[p][t])])]
ret	**mod**(N)!**qry**	**output**	ArgMax(EU)

We see that the function evaluates N different expected utilities, one for each decision option. ArgMax returns the option delivering the maximum expected utility.

In terms of Bayesian Decision Analysis, the outcome space X is all (predicted class (p), true class (t)) pairs, whose elements are given utilities by UPT. In the expected utility (EU) computations, the Action function only sums over the N outcomes that are consistent with the current p, that is, if the prediction is p, then the probability of any outcome (p',t) where $p' \neq p$ is 0 and can be dropped.

(3) F1 Score: Optimizing a more complex decision criterion We introduce another model, the Bayes Point Machine, and use it to illustrate a more complicated utility function, namely the F1 score. The F1 score is a measure of accuracy for binary classification that takes into account both false positives and false negatives.

table Data			
⋮			
ProbTrue	**real!qry**	**output**	**infer**.Bernoulli[].Bias(Y)
Train	**bool!det**	**input**	
table Ts			
Th	**real!det**	**input**	
Decisions	**bool**[SizeOf(Data)]	**output**	[**for** d < SizeOf(Data)→d.ProbTrue > Th]
ETP	**real!qry**	**output**	Sum([**for** d < SizeOf(Data)→
			if (!d.Train)& Decisions[d] **then** d.ProbTrue **else** 0.0])
EFP	**real!qry**	**output**	Sum([**for** d < SizeOf(Data)→
			if (!d.Train)& Decisions[d] **then** 1.0 − d.ProbTrue **else** 0.0])
EFN	**real!qry**	**output**	Sum([**for** d < SizeOf(Data)→
			if (!d.Train)& (!Decisions[d])**then** d.ProbTrue **else** 0.0])
EF1	**real!qry**	**output**	(2.0 ∗ ETP)/ ((2.0 ∗ ETP)+ EFP + EFN)
table Decisions			
ChosenThID	**link**(Ts)!**qry**	**static output**	ArgMax([**for** t < SizeOf(Ts)→t.EF1])
ChosenTh	**real!qry**	**static output**	ChosenThID.Th
DataID	**link**(Data)!**det**	**input**	
Decision	**bool!qry**	**output**	ChosenThID.Decisions[DataID]

As can be seen from the full Tabular code in Figure 3 (in the appendix), in table Data (abbreviated here), the data schema consists of seven real-valued clinical measurements X0 to X6 and a Boolean outcome variable Y to be predicted. The model represents a Bayes Point Machine [24] in which the prior over the weight vector W is drawn from a VectorGaussian, and the label Y is generated by thresholding a noisy score Z which is the inner product between the input vector and the weight vector W.

The set D of decision options is given in table Ts, which contains a number of possible thresholds Th for deciding on the test result depending on the marginal predictive probability for the label queried, per data point, by the expression **infer**.Bernoulli[].Bias (Y) in attribute ProbTrue. For each threshold, Decisions contains an array, indexed by data points, containing the candidate decision for each data point. The columns ETP, EFP, and EFN evaluate the expected number of true positives, false positives, and false negatives, respectively, by summing the relevant marginal probabilities over test data, which is valid due to linearity of the expectation operator. Finally, the approximate expected F1 score is calculated for each threshold using:

$$E[F_1] = E\left[\frac{2\cdot\mathrm{TP}}{2\cdot\mathrm{TP}+\mathrm{FP}+\mathrm{FN}}\right] \approx \frac{2\cdot E[\mathrm{TP}]}{2\cdot E[\mathrm{TP}]+E[\mathrm{FP}]+E[\mathrm{FN}]}\,.$$

This is an approximation because the F1 score is a non-linear function in TP, FP, and FN, and is employed here because it allows us to express the expectation in terms of marginal probabilities which are available from our inference back end. Recent work by Nowozin [27] has shown that approximations of this form yield good results.

Finally, the table Decisions determines the optimal threshold in the sense of expected F1 score and contains for each data point the labelling decision corresponding to the chosen threshold.

To sum up, the examples of this section illustrate how user-defined queries in Tabular can express the combination of a probabilistic model and a complex decision criterion in a succinct way in the context of the data schema.

7 Tabular Excel: Implementing Tabular in a Spreadsheet

Public releases of the Tabular add-in for Excel are available from `http://research.microsoft.com/tabular`. The add-in extends Excel with a new task pane for authoring models, running inference and setting parameters of Infer.NET. A user authors the model within a rectangular area of a worksheet. Tabular parses and type-checks the model in the background, enabling the inference button when the model is well-typed. Tabular pulls the data schema and data itself from the relational Data Model of Excel 2013. The results of inference and queries are then reported back to the user as augmented Excel tables. Tabular Excel is able to concisely express a wide range of models beyond those illustrated here (see Appendix B).

The implementation employs bi-directional type-checking to facilitate dependent typing. The space attribute of a type is optional in the concrete syntax and can be inferred from its definition. In indexed expressions, the second expression bounding the index is optional: it can typically be inferred from the **mod**-type of the indexing expression. Type checking the Tabular schema results in a type-annotated schema. This is elaborated to core form, eliminating all function calls and indexed models. The core schema is then translated to an Infer.NET [23] factor graph, constructed dynamically with Infer.NET's (imperative and weakly typed) modelling API. Our (type-directed) translation relies on and exploits the fact that all table sizes are known and that discrete random variables, which may be used to index into arrays, have known support. Moreover, the space of any (explicit or implicit) array indexing expression is used to insert the requisite Infer.NET *switch* construct when indexing through a **rnd**-space index (as demonstrated in Appendix C.)

The Infer.NET API, though used to construct factor graphs, can also be viewed as an API for constructing single-assignment, imperative programs with random expressions, fixed-size arrays, bounded for loops and conditional statements (but not expressions). Our compiler uses continuations to translate between the high-level functional language of Tabular and a lower-level imperative representation that is finally interpreted as calls to the Infer.NET API.

The fruits of Infer.NET inference are approximate marginal distributions for the **rnd**-space bindings of the schema. Expressions in **det** and **qry**-space are evaluated by interpretation after inference, binding input to the concrete data and **rnd**-level variables to their inferred distributions. Thus **qry**-space expressions have access to the inputs, deterministic values and distributions on which they depend. For compilation, the type system ensures that the value of **qry**-space expression cannot depend on the particular value of a **rnd**-space variable (only its distribution) and that all **rnd**-space variables can be inferred prior to **qry** evaluation.

Extracting Standalone Online Inference Code Our users can also extract C# or Excel-user-friendly Visual Basic source code to compile and run their models outside Excel. (Appendix C lists the extracted code for model Faithful.) This supports subsequent customization by Infer.NET experts as well as integration in standalone applications independent of Excel. One of our internal users has extracted code in this way to perform inference on a large dataset with approximately 42 million entities and 46 million relationships between them. Inference required 7.5 hours of processing time on a 2-core Intel Xenon L5640 server with 96 GB of RAM. The extracted code is also useful for debugging compilation and applications that need to separate learning (on training data) from prediction (on new data). Tabular Excel performs one-shot learning and prediction but incremental, online learning, a major advantage of Bayesian inference, appears to be a simple extension.

Experiments Figure 4 lists a selection of models run in Tabular Excel, along with data set sizes and run-times to indicate scale.

F1 Score for Mammography The following is direct comparison between the Tabular Excel form of the Mammography model (Figure 3) with code for the same problem written in C# using Infer.NET. We get the same statistical answers in both cases, though there are differences in code speed. Initially, Tabular queries were (naively) interpreted, not compiled; adopting simple runtime code generation techniques has allowed us to reduce the **qry** time from 1601ms (interpreted) to 29ms (compiled), a 55x fold increase. Simple optimizations like deforesting array summations may reduce times even further. The handwritten C# model is slower on inference because it is effectively compiled and run twice, once for training and another time for prediction.

	data (LOC)	model (LOC)	decisions (LOC)	inference (ms)	query (ms)
Infer.NET	35	35	45	2968	6
Tabular	0	15	14	1529	1601/29

The C# code expresses the core of the model quite succinctly in around 14 LOC (as Tabular does), but repeats it twice with only minor changes, once for the training data and once for prediction from the learned posterior. The C# code does two rounds of model construction and inference, once to infer the posterior on the training data, and another to make the predictions on the test data, which leads to more than twice the time spent in inference. The C# code for making decisions from the uncertain predictions is roughly 3 times as long as the Tabular code, but is compiled and run at native speed and thus 2 orders of magnitude faster than our **qry**-space interpreter but only 5 times faster than our most recent **qry** space compiler that dynamically generates and executes code.

TrueSkill applied to NCAA football

Model	Lang	LOC	Runtime (ms)	Model log evidence
TrueSkill	T	0 + 18 + 0	503	-491.95
TrueSkill	IN	60 + 24 + 18	381	-491.95
HomeAdv	T	0 + 21 + 0	663	-472.44
HomeAdv	IN	60 + 28 + 19	484	-472.44
Leagues	T	0 + 22 + 0	747	-445.81
Leagues	IN	60 + 40 + 23	537	-445.81

Leagues			
leagueId	**int!det**	**input**	
skillModifier	**real!rnd**	**output**	GaussianFromMeanAndPrecision(0.0,1.0)
Teams			
teamId	**int!det**	**input**	
leagueId	**mod**(SizeOf(Leagues))**!det**	**input**	
individualSkill	**real!rnd**	**output**	GaussianFromMeanAndPrecision(5.0,1.0)
skill	**real!rnd**	**output**	individualSkill + leagueId.skillModifier
Matches			
homeSkillAdvantage	**real!rnd**	**static output**	GaussianFromMeanAndPrecision(0.0,1.0)
gameId	**int!det**	**input**	
team1Id	**mod**(SizeOf(Teams))**!det**	**input**	
team1Score	**int!det**	**input**	
team2Id	**mod**(SizeOf(Teams))**!det**	**input**	
team2Score	**int!det**	**input**	
team1WasHome	**real!det**	**input**	
team1HomeAdvantage	**real!rnd**	**output**	team1WasHome $*$ homeSkillAdvantage
team1Perf	**real!rnd**	**output**	GaussianFromMeanAndPrecision(
			team1Id.skill + team1HomeAdvantage,1.0)
team2Perf	**real!rnd**	**output**	GaussianFromMeanAndPrecision(
			team2Id.skill,1.0)
team1Won	**bool!rnd**	**output**	team1Perf > team2Perf

Fig. 1. Tabular NCAA_TrueSkill model.

We use Tabular to recreate the model described in Tarlow et al. [36] and apply it to NCAA Football (NCAAF) data from the 2013 season, which includes 845 matches between 206 teams from 12 leagues. The model is an extended version of TrueSkill that incorporates home field advantage and league information: these are modelled by introducing latent variables that correspond to the home field advantage, and a separate variable for each league representing the league's strength. A team's skill in a particular game is then assumed to be a sum of its individual skill, the skill of the league that it is in, and (if the team is the home team) the globally shared home field advantage. Beyond these differences, the model is equivalent to TrueSkill.

We also implemented two simplified versions of the full model. In the first, we eliminated the league-specific skills; in the second, we also eliminated the home field advantage, which makes the model equivalent to standard TrueSkill. As a baseline, we implemented the models in C# using Infer.NET directly. Figure 2 depicts the C# code to construct the NCAA model, without the additional code required to read the data and perform inference. In the previous table, we compare the Tabular implementations to the C# implementations. In all three cases, the results of inference are identical in terms of the inferred posteriors and the model evidence.[1] A comparison of lines of code needed to read the data, express the model and invoke inference by observing data is also included, showing that the Tabular implementation allows the user to focus on the task of expressing the model, leaving data manipulation to the runtimes. Although the C# modeling code is not terribly long, it is quite obscure and requires relatively sophisticated knowledge of C# generics and careful use of an imperative modeling API.

[1] The *model evidence* or *marginal likelihood* is the probability of the observed output data under the model. This quantity is commonly used to compare alternative models of the same data; higher quantities correspond to better models.

```
public void InitializeModel_TrueSkill (int numTeams, int numLeagues) {
   evidence = Variable.Bernoulli (0.5).Named("evidence");
   block = Variable.If (evidence);
   nGames = Variable.New<int>().Named("nGames");
   nTeams = Variable.New<int>().Named("nTeams");
   nLeagues = Variable.New<int>().Named("nLeagues");
   team = new Range(nTeams).Named("team");
   game = new Range(nGames).Named("game");
   league = new Range(nLeagues).Named("league");
   var individualSkills   = Variable.Array<double>(team).Named("individualSkills");
   leagueSkills  = Variable.Array<double>(league).Named("leagueSkills");
   skills  = Variable.Array<double>(team).Named("skills");
   teamToLeague = Variable.Array<int>(team).Named("teamToLeague");
   team1Id = Variable.Array<int>(game).Named("team1Id");
   team2Id = Variable.Array<int>(game).Named("team2Id");
   team1LeagueId = Variable.Array<int>(game).Named("team1LeagueId");
   team2LeagueId = Variable.Array<int>(game).Named("awayLeagueIdx");
   team1Won = Variable.Array<bool>(game).Named("team1Won");
   team1WasHome = Variable.Array<double>(game).Named("team1WasHome");
   var team1Skill = Variable.Array<double>(game).Named("team1Skill");
   var team2Skill = Variable.Array<double>(game).Named("team2Skill");
   var league1Skill  = Variable.Array<double>(game).Named("league1Skill");
   var league2Skill  = Variable.Array<double>(game).Named("league2Skill");
   var team1HomeAdvantage = Variable.Array<double>(game).Named("team1HomeAdvantage");
   team1Perf = Variable.Array<double>(game).Named("team1Perf");
   team2Perf = Variable.Array<double>(game).Named("team2Perf");
   homeSkillAdvantage = Variable.GaussianFromMeanAndPrecision(0.0, 1.0);
   leagueSkills [league] = Variable.GaussianFromMeanAndPrecision(0.0, 1.0).ForEach(league);
   individualSkills  [team] = Variable.GaussianFromMeanAndPrecision(5.0, 1.0).ForEach(team)
        ;
   using (Variable.ForEach(team)) {
      skills [team] = individualSkills [team] + leagueSkills[teamToLeague[team]];
   }
   using (Variable.ForEach(game)){
      team1HomeAdvantage[game] = team1WasHome[game] * homeSkillAdvantage;
      team1Perf[game] = Variable.GaussianFromMeanAndPrecision(skills[team1Id[game]] +
           team1HomeAdvantage[game], 1.0);
      team2Perf[game] = Variable.GaussianFromMeanAndPrecision(skills[team2Id[game]],
           1.0);
      team1Won[game] = (team1Perf[game] > team2Perf[game]);
   }
   block.CloseBlock();
}
```

Fig. 2. Infer.NET NCAA_TrueSkill model, excluding data input, observation and inference code (compare to Figure 1).

8 Related Work

Interest in probabilistic programming languages is rising as evinced by recent languages like Church [13], a Turing-complete probabilistic Scheme with inference based on sampling, and its relatives Anglican [37] (a typed re-imagination of Church) and Venture [21] (a variant of Church offering programmable inference). Other recent works include R2 [26], which uses program analysis to optimize MCMC sampling of probabilistic programming, and Uncertain<T> [5], a simple abstraction for embedding probabilistic reasoning into conventional programs that handle uncertain data.

To the best of our knowledge, few systems offer explicit support for decision theory. IBAL's [29] impressive framework aims to combine Bayesian inference and decision theory "under a single coherent semantic framework". IBAL makes use of query information and only computes quantities needed to answer specific queries. Other systems that extend probabilistic languages with dedicated decision theoretic constructs are described in [8, 6, 25]. The main difference in our approach is that while our post-processing can be used to implement decision theory strategies, decision theoretic constructs are not built into the language. This is a pragmatic choice. In general, decision theory involves two intractabilities: computing expected utilities, and optimizing over the decisions. IBAL and DT-ProbLog [6] have some general-purpose approximations, but often problem-specific approximations are needed as in our F1 optimization example or in [27]. It is not clear how these approximations fit into the above frameworks. Tabular's free-form post-processing design allows such bespoke approximations.

STAN [35] allows for post-processing of inference results, but only via separately declared code blocks, rather than being conveniently mingled with the model or abstracted in functions. Although STAN's facilities are expressive and can include arbitrary deterministic and stochastic computations, they are restricted to computing *per sample* quantities. In Tabular terms, this would correspond to computations restricted to **rnd**-space which prevents the computation of the aggregate **qry**-space quantities required for Bayesian decision theory.

Figaro [30] supports post-inference decision-making, but via separate, decision-specific language features, outside the core modelling language. Tabular, instead, uses types to distinguish between operations available in different spaces (or phases) (such as random draws in **rnd** space, optimization (ArgMax) and moments of distributions in **qry**-space). Embedded DSLs such as Infer.NET [23], HANSEI [19] and FACTORIE [22] enable arbitrary post-processing in the host language, but require knowledge of both the host and the embedded language, which is typically much simpler.

Tabular is, to our knowledge, the first probabilistic programming language with dependently typed abstractions. STAN and BUGS [11] do have value-indexed types, but cannot abstract over indexes appearing in types.

We advocate types to help catch errors in probabilistic queries on spreadsheets. There is a body of work on testing and discovering errors on spreadsheets. For example, Ahmad et al. [1] propose unit-based types as a means of catching errors. To the best of our knowledge, dependent-types have not previously been applied to spreadsheets.

Tabular was directly inspired by the question of finding a textual notation for the factor graphs generated by InfernoDB [34] which constructs a hierarchical mixture-based graphical model in Infer.NET [23] from an arbitrary relational schema. CrossCat

[33] is a related model, which handles single tables with mixed types (real, integer, bool).

Finally, we list the main improvements in the present design of Tabular, compared to the original publication [16].

- Determinacy annotations to help with compilation to Infer.NET.
- Just two levels, not three. Change of terminology to **static/inst** instead of h/w/xyz.
- Local variables subject to alpha-equivalence.
- Self-contained semantics at the Tabular level, ie, function types are tables too.
- Query expressions to determine how results of inference are returned (generalizing query-by-latent-column and query-by-missing value), and now supporting decision-making under uncertainty.
- Function definitions and function application for easy reuse of common idioms (generalizing fixed set of primitive models).
- A system of dependent types in particular, sizes of arrays and integer ranges, to detect certain errors and to streamline the syntax of indexed models.
- Implementation in spreadsheet (Microsoft Excel) rather than a database (Microsoft Access).

9 Conclusions

We recast Tabular as a query language on databases held in spreadsheets.

We added user-defined functions with statically-checked value-dependent types. We described its formal semantics and an implementation in Excel. We reported results on a range of examples, including demonstrating that some decision-theoretic problems previously solved by C# code can be solved by compact annotations in a spreadsheet. Our work brings probabilistic programming to spreadsheet users with some knowledge of statistics, but without needing them to write code beyond spreadsheet annotations.

This paper presents a technical evaluation of the design consisting of theorems about its metatheory, demonstration of its expressiveness by example, and some numeric comparisons with the alternative of writing models directly in Infer.NET. Evaluating the usability by spreadsheet users is important, but we leave that task for future work.

We have in mind several lines of future development. One limitation of our current system is that data is modelled by map-style loops over data; to model time-series, it would be useful to add some form of iterative fold-style loops. Another limitation is that programs involve a single run of the underlying inference system: **rnd**-space determines the model and its conditioning, and **qry**-space determines how the results are processed. To support multiple runs of inference we might consider an indexed hierarchy of spaces where **infer** moves data from $\textbf{rnd}_i$ space to $\textbf{qry}_i$ space, and $\textbf{rnd}_{i+1}$ space can depend on results computed in $\textbf{qry}_i$ space.

Finally, our approach could be applied to add user-defined functions to languages such as BUGS or Stan, or to design typed-forms of universal probabilistic languages such as those in the Church family.

Acknowledgement Dylan Hutchison commented on a draft. We thank Natalia Larios Delgado and Matthew Smith for their feedback on our Excel addin.

References

[1] Ahmad, Y., Antoniu, T., Goldwater, S., Krishnamurthi, S.: A type system for statically detecting spreadsheet errors. In: 18th IEEE International Conference on Automated Software Engineering (ASE 2003), 6-10 October 2003, Montreal, Canada. pp. 174–183 (2003)

[2] Billingsley, P.: Probability and Measure. Wiley, 3rd edn. (1995)

[3] Borgström, J., Gordon, A.D., Greenberg, M., Margetson, J., Gael, J.V.: Measure transformer semantics for Bayesian machine learning. Logical Methods in Computer Science 9(3) (2013)

[4] Borgström, J., Gordon, A.D., Greenberg, M., Margetson, J., Van Gael, J.: Measure transformer semantics for Bayesian machine learning. In: European Symposium on Programming (ESOP'11). LNCS, vol. 6602, pp. 77–96. Springer (2011)

[5] Bornholt, J., Mytkowicz, T., McKinley, K.S.: Uncertain<T>: A first-order type for uncertain data. In: Architectural Support for Programming Languages and Operating Systems (ASPLOS) (March 2014), `http://research.microsoft.com/apps/pubs/default.aspx?id=208236`

[6] Van den Broeck, G., Thon, I., van Otterlo, M., De Raedt, L.: DTProbLog: A decision-theoretic probabilistic Prolog. In: AAAI (2010)

[7] Cardelli, L.: Typeful programming. Tech. Rep. 52, Digital SRC (1989)

[8] Chen, J., Muggleton, S.: Decision-theoretic logic programs. In: Proceedings of ILP. p. 136 (2009)

[9] Duda, R.O., Hart, P.E.: Pattern Classification and Scene Analysis. John Wiley & Sons, New York, NY (1973)

[10] Gelman, A., Carlin, J.B., Stern, H.S., Dunson, D.B., Vehtari, A., Rubin, D.B.: Bayesian Data Analysis. Chapman & Hall, 3rd edn. (2014)

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

[12] Giry, M.: A categorical approach to probability theory. In: Banaschewski, B. (ed.) Categorical Aspects of Topology and Analysis, Lecture Notes in Mathematics, vol. 915, pp. 68–85. Springer (1982)

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

[14] Goodman, N.D.: The principles and practice of probabilistic programming. In: Principles of Programming Languages (POPL'13). pp. 399–402 (2013)

[15] Gordon, A.D., Aizatulin, M., Borgström, J., Claret, G., Graepel, T., Nori, A., Rajamani, S., Russo, C.: A model-learner pattern for Bayesian reasoning. In: POPL (2013)

[16] Gordon, A.D., Graepel, T., Rolland, N., Russo, C.V., Borgström, J., Guiver, J.: Tabular: a schema-driven probabilistic programming language. In: POPL (2014)

[17] Gordon, A.D., Henzinger, T.A., Nori, A.V., Rajamani, S.K.: Probabilistic programming. In: Future of Software Engineering (FOSE 2014). pp. 167–181 (2014)

[18] Herbrich, R., Minka, T., Graepel, T.: TrueSkill$^{\text{tm}}$: A Bayesian skill rating system. In: Advances in Neural Information Processing Systems (NIPS'06) (2006)

[19] Kiselyov, O., Shan, C.: Embedded probabilistic programming. In: Conference on Domain-Specific Languages. pp. 360–384 (2009)

[20] Kozen, D.: Semantics of probabilistic programs. Journal of Computer and System Sciences 22(3), 328–350 (1981)

[21] Mansinghka, V., Selsam, D., Perov, Y.: Venture: a higher-order probabilistic programming platform with programmable inference. arXiv preprint arXiv:1404.0099 (2014)

[22] McCallum, A., Schultz, K., Singh, S.: Factorie: Probabilistic programming via imperatively defined factor graphs. In: Advances in Neural Information Processing Systems (NIPS'09). pp. 1249–1257 (2009)

[23] Minka, T., Winn, J., Guiver, J., Knowles, D.: Infer.NET 2.5 (2012), microsoft Research Cambridge. http://research.microsoft.com/infernet

[24] Minka, T.P.: A family of algorithms for approximate Bayesian inference. Ph.D. thesis, Massachusetts Institute of Technology (2001)

[25] Nath, A., Domingos, P.: A language for relational decision theory. In: Proceedings of the International Workshop on Statistical Relational Learning (2009)

[26] Nori, A.V., Hur, C.K., Rajamani, S.K., Samuel, S.: R2: An efficient MCMC sampler for probabilistic programs. In: AAAI Conference on Artificial Intelligence (AAAI). AAAI (July 2014)

[27] Nowozin, S.: Optimal decisions from probabilistic models: the intersection-over-union case. In: Proceedings of CVPR 2014 (2014)

[28] Panangaden, P.: Labelled Markov processes. Imperial College Press (2009)

[29] Pfeffer, A.: The design and implementation of IBAL: A general-purpose probabilistic language. In: Getoor, L., Taskar, B. (eds.) Introduction to Statistical Relational Learning. MIT Press (2007)

[30] Pfeffer, A.: Figaro: An object-oriented probabilistic programming language. Tech. rep., Charles River Analytics (2009)

[31] Pitts, A.M.: Nominal Sets: Names and Symmetry in Computer Science, Cambridge Tracts in Theoretical Computer Science, vol. 57. Cambridge University Press (2013)

[32] Ramsey, N., Pfeffer, A.: Stochastic lambda calculus and monads of probability distributions. In: POPL. pp. 154–165 (2002)

[33] Shafto, P., Kemp, C., Mansinghka, V., Gordon, M., Tenenbaum, J.B.: Learning cross-cutting systems of categories. In: Proceedings of the 28th Annual Conference of the Cognitive Science Society (2006)

[34] Singh, S., Graepel, T.: Compiling relational database schemata into probabilistic graphical models. CoRR abs/1212.0967 (2012)

[35] Stan Development Team: Stan: A C++ library for probability and sampling, version 2.2 (2014), `http://mc-stan.org/`

[36] Tarlow, D., Graepel, T., Minka, T.: Knowing what we don't know in NCAA Football ratings: Understanding and using structured uncertainty. In: MIT Sloan Sports Analytics Conference (2014)

[37] Wood, F., van de Meent, J.W., Mansinghka, V.: A new approach to probabilistic programming inference. In: Proceedings of the 17th International conference on Artificial Intelligence and Statistics (2014)

[38] Xi, H., Pfenning, F.: Eliminating array bound checking through dependent types. In: Proceedings of the ACM SIGPLAN '98 Conference on Programming Language Design and Implementation (PLDI). pp. 249–257 (1998)

table Data			
X0	real!**det**	**input**	
X1	real!**det**	**input**	
X2	real!**det**	**input**	
X3	real!**det**	**input**	
X4	real!**det**	**input**	
X5	real!**det**	**input**	
X6	real!**det**	**input**	
Mean	vector!**det**	**static output**	VectorFromArray([**for** i < 7 →0.0])
CoVar	PositiveDefiniteMatrix!**det**	**static output**	IdentityScaledBy(7,1.0)
W	vector!**rnd**	**static output**	VectorGaussianFromMeanAndVariance(Mean,CoVar)
Noise	real!**det**	**static output**	0.1
Z	real!**rnd**	**output**	InnerProduct(W,VectorFromArray([X0;X1;X2;X3;X4;X5;X6]))
Y	bool!**rnd**	**output**	Gaussian(Z,Noise)> 0.0
ProbTrue	real!**qry**	**output**	**infer**.Bernoulli[].Bias(Y)
Train	bool!**det**	**input**	
table Ts			
Th	real!**det**	**input**	
Decisions	bool[SizeOf(Data)]	**output**	[**for** d < SizeOf(Data)→d.ProbTrue > Th]
ETP	real!**qry**	**output**	Sum([**for** d < SizeOf(Data)→
			if (!d.Train)& Decisions[d] **then** d.ProbTrue **else** 0.0])
EFP	real!**qry**	**output**	Sum([**for** d < SizeOf(Data)→
			if (!d.Train)& Decisions[d] **then** 1.0 − d.ProbTrue **else** 0.0])
EFN	real!**qry**	**output**	Sum([**for** d < SizeOf(Data)→
			if (!d.Train)& (!Decisions[d])**then** d.ProbTrue **else** 0.0])
EF1	real!**qry**	**output**	(2.0 ∗ ETP)/ ((2.0 ∗ ETP)+ EFP + EFN)
table Decisions			
ChosenThID	link(Ts)!**qry**	**static output**	ArgMax([**for** t < SizeOf(Ts)→t.EF1])
ChosenTh	real!**qry**	**static output**	ChosenThID.Th
DataID	link(Data)!**det**	**input**	
Decision	bool!**qry**	**output**	ChosenThID.Decisions[DataID]

Fig. 3. F1 computation in Tabular on mammography data (complete model)

A Functions of a Standard Prelude

A.1 Conjugate Bernoulli

fun CBernoulli			
hAlpha	**real!det**	**static input**	
hBeta	**real!det**	**static input**	
Bias	**real!rnd**	**static output**	Beta(hAlpha,hBeta)
B	**bool!rnd**	**output**	Bernoulli(Bias)

Type:

fun CBernoulli		
hAlpha	**real!det**	**static input**
hBeta	**real!det**	**static input**
Bias	**real!rnd**	**static output**
B	**bool!rnd**	**output**

A.2 Conjugate Discrete

fun CDiscrete			
N	**int!det**	**static input**	
alpha	**real!rnd**	**static input**	
V	**real**[N]**!rnd**	**static output**	Dirichlet[N]([**for** i $<$ N $\rightarrow$ alpha])
D	**mod** N**!rnd**	**output**	Discrete(V)

Type:

fun CDiscrete		
N	**int!det**	**static input**
alpha	**real!rnd**	**static input**
V	**real**[N]**!rnd**	**static output**
D	**mod** N**!rnd**	**output**

A.3 Conjugate Gaussian

fun CGaussian			
hMean	**real!det**	**static input**	
hPrec	**real!det**	**static input**	
hShape	**real!det**	**static input**	
hScale	**real!det**	**static input**	
Mean	**real!rnd**	**static output**	Gaussian(hMean,hPrec)
Prec	**real!rnd**	**static output**	Gamma(hShape,hScale)
G	**real!rnd**	**output**	Gaussian(Mean,Prec)

Type:

fun CGaussian		
hMean	**real!det**	**static input**
hPrec	**real!det**	**static input**
hShape	**real!det**	**static input**
hScale	**real!det**	**static input**
Mean	**real!rnd**	**static output**
Prec	**real!rnd**	**static output**
G	**real!rnd**	**output**

A.4 Conjugate VectorGaussian

Our implementation contains a standard prelude of function definitions. To illustrate, we give here the more complicated definition of a conjugate n-dimensional multivariate gaussian, whose prior is an n-dimensional vector for the mean (a vector) and a (positive definite) $(n \times n)$-matrix for the covariance, with a Wishart prior (a multidimensional Gamma distribution):

fun CVectorGaussian			
N	**int!det**	**static input**	
hMeanVectorPrecisionCount	**real!det**	**static input**	
hWishartShapeConstant	**real!det**	**static input**	
hWishartScaleConstant	**real!det**	**static input**	
meanVector	**real**[N]**!det**	**static local**	[**for** i < N → 0.0]
precisionMatrix	**real**[N][N]**!det**	**static local**	IdentityScaledBy(N, hMeanVectorPrecisionCount)
mean	**real**[N]**!rnd**	**static output**	VectorGaussian(meanVector,precisionMatrix)
scale	**real**[N][N]**!rnd**	**static local**	IdentityScaledBy(N,hWishartScaleConstant)
covariance	**real**[N][N]**!rnd**	**static output**	WishartFromShapeAndScale(hWishartShapeConstant, scale)
y	**real**[N]**!rnd**	**output**	VectorGaussian(mean,covariance)

The function above has the function type below.

N	**int!det**	**static input**
hMeanVectorPrecisionCount	**real!det**	**static input**
hWishartShapeConstant	**real!det**	**static input**
hWishartScaleConstant	**real!det**	**static input**
mean	**real**[N]**!rnd**	**static output**
covariance	**real**[N][N]**!rnd**	**static output**
y	**real**[N]**!rnd**	**output**

B Examples

Tabular Excel is able to concisely express a wide range of models beyond those illustrated here, including the relational DARE crowdsourcing model described in Gordon et al. [16], hierarchical and multivariate linear regression, various recommenders (Block Recommender,MatchBox, PCA), Latent Dirichlet Allocation (LDA) and classic classifiers (NaiveBayes, BPM, Mammography). Figure 4 summarizes a selection of models, with salient statistics.

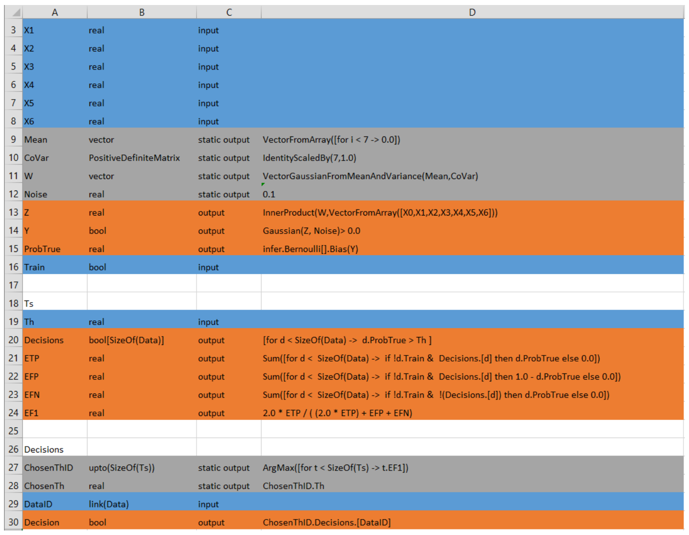
Figures 5 and 6 show our mammography model making decisions on the mammography dataset (with 11183 rows).

C Example: Extracted C# code for Faithful

The Tabular generated code for the Old Faithful example reveals the concision of Tabular models over Infer.NET ones (though a handwritten model would not be in A-normal form and thus somewhat shorter). Note that the model is constructed imperatively using C# using blocks to indicate the opening and closing of scopes for loops and conditionals in the model. This code also illustrates and the various pragmas asserting the ranges (sizes) of arrays, value ranges (bounds) of integers, and special Switch annotations, all of which are necessary and inserted by our type-directed translation from Tabular (see coloured occurrences of Range, SetValueRange and Switch below). While the method

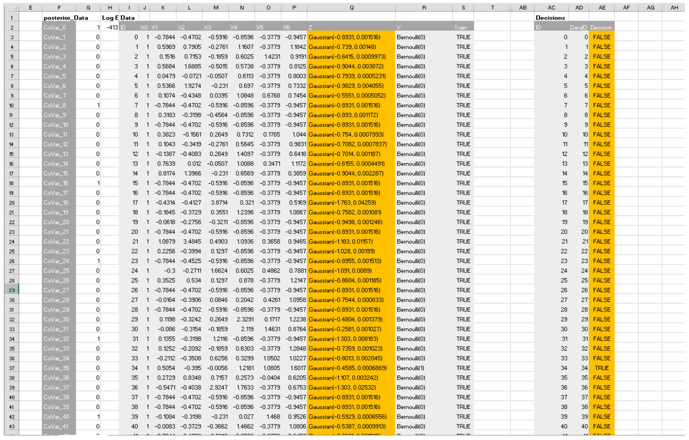
model	rows	cells	compile ms	inference ms	query ms	algorithm	iterations
DARE	12 565	34 465	301	11 722	0 / 0	EP	30
TrueSkill	20 100	60 100	5	738	0 / 0	EP	30
TrueSkill	2 010 000	6 010 000	97	43191	0 / 0	EP	30
TrueSkillBets	21 100	61 100	9	735	17 / 9	EP	30
Recommender	710	1130	50	5715	0 / 0	EP	30
NCAAF	1069	3826	113	495	0 / 0	EP	30
PCA	37	111	7	1063	0 / 0	EP	30
BPM	37	37	5	268	0 / 0	EP	30
LinearRegression	34	34	9	207	0 / 0	EP	30
Faithful	272	816	111	856	0 / 0	VMP	50
Matchbox	10 923	56 006	58	65968	0 / 0	EP	30
MammographyBPM	11 183	11 183	3	1586	0 / 0	EP	30
MammographyF1	22 386	22 366	11	1529	1601 / 29	EP	30
LDA	622	1 047	100	1791	0 / 0	EP	30
NaiveBayesQry	9	45	26	855	0 / 1	VMP	50

Fig. 4. Tabular examples: rows is sum of table sizes; cells is number of cells in output; compile time is elaboration and translation to Infer.NET; inference time is Infer.NET compilation and inference time; query time is post-processing of **qry**-level expressions (interpreted/compiled); all times in ms. (machine config.: DELL Precision T3600, Intel(R) Xeon(R) CPU E5-1620 with 16GB RAM, Windows 8 Enterprise and .NET 4.0). The MammographyF1 example shows a 55x speedup from adopting JIT-compiled **qry**-level processing.



	A	B	C	D
3	X1	real	input	
4	X2	real	input	
5	X3	real	input	
6	X4	real	input	
7	X5	real	input	
8	X6	real	input	
9	Mean	vector	static output	VectorFromArray([for i < 7 -> 0.0])
10	CoVar	PositiveDefiniteMatrix	static output	IdentityScaledBy(7,1.0)
11	W	vector	static output	VectorGaussianFromMeanAndVariance(Mean,CoVar)
12	Noise	real	static output	0.1
13	Z	real	output	InnerProduct(W,VectorFromArray([X0,X1,X2,X3,X4,X5,X6]))
14	Y	bool	output	Gaussian(Z, Noise)> 0.0
15	ProbTrue	real	output	infer.Bernoulli[].Bias(Y)
16	Train	bool	input	
17				
18	Ts			
19	Th	real	input	
20	Decisions	bool[SizeOf(Data)]	output	[for d < SizeOf(Data) -> d.ProbTrue > Th]
21	ETP	real	output	Sum([for d < SizeOf(Data) -> if !d.Train & Decisions.[d] then d.ProbTrue else 0.0])
22	EFP	real	output	Sum([for d < SizeOf(Data) -> if !d.Train & Decisions.[d] then 1.0 - d.ProbTrue else 0.0])
23	EFN	real	output	Sum([for d < SizeOf(Data) -> if !d.Train & !(Decisions.[d]) then d.ProbTrue else 0.0])
24	EF1	real	output	2.0 * ETP / ((2.0 * ETP) + EFP + EFN)
25				
26	Decisions			
27	ChosenThID	upto(SizeOf(Ts))	static output	ArgMax([for t < SizeOf(Ts) -> t.EF1])
28	ChosenTh	real	static output	ChosenThID.Th
29	DataID	link(Data)	input	
30	Decision	bool	output	ChosenThID.Decisions.[DataID]

Fig. 5. Mammography Model



	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	AB	AC	AD	AE	AF	AG	AH
1		posterior_Data		Log E	Data													Decisions					
2		CoVar_0	1	-413	ID	X0	X1	X2	X3	X4	X5	X6	Z	Y	Train			ID	DataID	Decision			
3		CoVar_1	0		0	1	-0.7844	-0.4702	-0.5916	-0.8596	-0.3779	-0.9457	Gaussian(-0.8931, 0.001516)	Bernoulli(0)	TRUE			0	0	FALSE			
4		CoVar_2	0		1	1	0.5969	0.7905	-0.2761	1.1607	-0.3779	1.1842	Gaussian(-0.739, 0.00148)	Bernoulli(0)	TRUE			1	1	FALSE			
5		CoVar_3	0		2	1	0.1516	0.7153	-0.1859	0.6025	1.4231	0.9191	Gaussian(-0.6415, 0.0009973)	Bernoulli(0)	TRUE			2	2	FALSE			
6		CoVar_4	0		3	1	0.5684	1.6885	-0.5015	0.5738	-0.3779	0.8125	Gaussian(-0.9044, 0.003872)	Bernoulli(0)	TRUE			3	3	FALSE			
7		CoVar_5	0		4	1	0.0479	-0.0721	-0.0507	0.6113	-0.3779	0.8003	Gaussian(-0.7939, 0.0005231)	Bernoulli(0)	TRUE			4	4	FALSE			
8		CoVar_6	0		5	1	0.5366	1.9274	-0.231	0.697	-0.3779	0.7332	Gaussian(-0.9829, 0.004055)	Bernoulli(0)	TRUE			5	5	FALSE			
9		CoVar_7	0		6	1	0.1074	-0.4348	0.0395	1.0848	0.6768	0.7454	Gaussian(-0.5551, 0.0005052)	Bernoulli(0)	TRUE			6	6	FALSE			
10		CoVar_8	1		7	1	-0.7844	-0.4702	-0.5916	-0.8596	-0.3779	-0.9457	Gaussian(-0.8931, 0.001516)	Bernoulli(0)	TRUE			7	7	FALSE			
11		CoVar_9	0		8	1	0.3183	-0.3198	-0.4564	-0.8596	-0.3779	-0.9457	Gaussian(-0.893, 0.001172)	Bernoulli(0)	TRUE			8	8	FALSE			
12		CoVar_10	0		9	1	-0.7844	-0.4702	-0.5916	-0.8596	-0.3779	-0.9457	Gaussian(-0.8931, 0.001516)	Bernoulli(0)	TRUE			9	9	FALSE			
13		CoVar_11	0		10	1	0.3823	-0.1561	0.2649	0.7312	0.1765	1.044	Gaussian(-0.754, 0.0007993)	Bernoulli(0)	TRUE			10	10	FALSE			
14		CoVar_12	0		11	1	0.1043	-0.3419	-0.2761	0.5645	-0.3779	0.9831	Gaussian(-0.7082, 0.0007837)	Bernoulli(0)	TRUE			11	11	FALSE			
15		CoVar_13	0		12	1	-0.1387	-0.4083	0.2649	1.4097	-0.3779	0.6418	Gaussian(-0.7014, 0.001187)	Bernoulli(0)	TRUE			12	12	FALSE			
16		CoVar_14	0		13	1	0.7639	0.012	-0.0507	1.0088	0.3471	1.1172	Gaussian(-0.6155, 0.0004491)	Bernoulli(0)	TRUE			13	13	FALSE			
17		CoVar_15	0		14	1	0.8174	1.3966	-0.231	0.6569	-0.3779	0.3859	Gaussian(-0.9044, 0.002287)	Bernoulli(0)	TRUE			14	14	FALSE			
18		CoVar_16	1		15	1	-0.7844	-0.4702	-0.5916	-0.8596	-0.3779	-0.9457	Gaussian(-0.8931, 0.001516)	Bernoulli(0)	TRUE			15	15	FALSE			
19		CoVar_17	0		16	1	-0.7844	-0.4702	-0.5916	-0.8596	-0.3779	-0.9457	Gaussian(-0.8931, 0.001516)	Bernoulli(0)	TRUE			16	16	FALSE			
20		CoVar_18	0		17	1	-0.4314	-0.4127	3.8714	0.321	-0.3779	0.5169	Gaussian(-1.763, 0.04259)	Bernoulli(0)	TRUE			17	17	FALSE			
21		CoVar_19	0		18	1	-0.1845	-0.3729	0.3551	1.2396	-0.3779	1.0867	Gaussian(-0.7582, 0.001081)	Bernoulli(0)	TRUE			18	18	FALSE			
22		CoVar_20	0		19	1	-0.0618	-0.2756	-0.3211	-0.8596	-0.3779	-0.9457	Gaussian(-0.9498, 0.001246)	Bernoulli(0)	TRUE			19	19	FALSE			
23		CoVar_21	0		20	1	-0.7844	-0.4702	-0.5916	-0.8596	-0.3779	-0.9457	Gaussian(-0.8931, 0.001516)	Bernoulli(0)	TRUE			20	20	FALSE			
24		CoVar_22	0		21	1	1.0879	3.4845	0.4903	1.0936	0.3658	0.9465	Gaussian(-1.183, 0.01157)	Bernoulli(0)	TRUE			21	21	FALSE			
25		CoVar_23	0		22	1	0.2256	-0.3994	0.1297	-0.8596	-0.3779	-0.9457	Gaussian(-1.028, 0.00199)	Bernoulli(0)	TRUE			22	22	FALSE			
26		CoVar_24	1		23	1	-0.7844	-0.4525	-0.5916	-0.8596	-0.3779	-0.9457	Gaussian(-0.8955, 0.001513)	Bernoulli(0)	TRUE			23	23	FALSE			
27		CoVar_25	0		24	1	-0.3	-0.2711	1.6624	0.6025	0.4862	0.7881	Gaussian(-1.091, 0.0089)	Bernoulli(0)	TRUE			24	24	FALSE			
28		CoVar_26	0		25	1	0.3525	0.534	0.1297	0.878	-0.3779	1.2147	Gaussian(-0.8604, 0.001185)	Bernoulli(0)	TRUE			25	25	FALSE			
29		CoVar_27	0		26	1	-0.7844	-0.4702	-0.5916	-0.8596	-0.3779	-0.9457	Gaussian(-0.8931, 0.001516)	Bernoulli(0)	TRUE			26	26	FALSE			
30		CoVar_28	0		27	1	-0.0164	-0.3906	0.0846	0.2042	0.4261	1.0958	Gaussian(-0.7544, 0.000833)	Bernoulli(0)	TRUE			27	27	FALSE			
31		CoVar_29	0		28	1	-0.7844	-0.4702	-0.5916	-0.8596	-0.3779	-0.9457	Gaussian(-0.8931, 0.001516)	Bernoulli(0)	TRUE			28	28	FALSE			
32		CoVar_30	0		29	1	0.1198	-0.3242	0.2649	2.3291	0.1717	1.2238	Gaussian(-0.4804, 0.001379)	Bernoulli(0)	TRUE			29	29	FALSE			
33		CoVar_31	0		30	1	-0.086	-0.3154	-0.1859	2.119	1.4631	0.8764	Gaussian(-0.2581, 0.001027)	Bernoulli(0)	TRUE			30	30	FALSE			
34		CoVar_32	1		31	1	0.1355	-0.3198	1.2116	-0.8596	-0.3779	-0.9457	Gaussian(-1.303, 0.008163)	Bernoulli(0)	TRUE			31	31	FALSE			
35		CoVar_33	0		32	1	0.1252	-0.2092	-0.1859	0.6303	-0.3779	1.2848	Gaussian(-0.7359, 0.001023)	Bernoulli(0)	TRUE			32	32	FALSE			
36		CoVar_34	0		33	1	-0.2112	-0.3508	0.6256	0.3299	1.0502	1.0227	Gaussian(-0.8013, 0.002045)	Bernoulli(0)	TRUE			33	33	FALSE			
37		CoVar_35	0		34	1	0.5054	-0.395	-0.0056	1.2181	1.0805	1.6017	Gaussian(-0.4585, 0.0006869)	Bernoulli(1)	TRUE			34	34	TRUE			
38		CoVar_36	0		35	1	0.2729	0.8348	0.7157	0.2573	-0.0404	0.6205	Gaussian(-1.107, 0.003242)	Bernoulli(0)	TRUE			35	35	FALSE			
39		CoVar_37	0		36	1	-0.5471	-0.4038	2.9247	1.7633	-0.3779	0.6753	Gaussian(-1.303, 0.02532)	Bernoulli(0)	TRUE			36	36	FALSE			
40		CoVar_38	0		37	1	-0.7844	-0.4702	-0.5916	-0.8596	-0.3779	-0.9457	Gaussian(-0.8931, 0.001516)	Bernoulli(0)	TRUE			37	37	FALSE			
41		CoVar_39	0		38	1	-0.7844	-0.4702	-0.5916	-0.8596	-0.3779	-0.9457	Gaussian(-0.8931, 0.001516)	Bernoulli(0)	TRUE			38	38	FALSE			
42		CoVar_40	1		39	1	-0.1084	-0.3198	-0.231	0.027	1.468	0.9526	Gaussian(-0.5929, 0.0006556)	Bernoulli(0)	TRUE			39	39	FALSE			
43		CoVar_41	0		40	1	-0.0083	-0.3729	-0.3662	1.4662	-0.3779	1.0806	Gaussian(-0.5387, 0.0009913)	Bernoulli(0)	TRUE			40	40	FALSE			

Fig. 6. Mammography Results

is large, its signature is small and fairly easy to call from client code: it takes sizes, input and output columns of the database as (in or out) parameters, one column at a time, in schema order. (The typing of outputs could be tightened up further still.)

```
public static void Infer(MicrosoftResearch.Infer.IAlgorithm _algorithm, int _iterations ,
    out Bernoulli  out_evidence ,
    int in_faithful_size   ,
    out object out_faithful_cluster_V  , out object out_faithful_cluster  ,
    out object out_faithful_duration_Mean, out object out_faithful_duration_Prec ,
    double?[] in_faithful_duration  , out object out_faithful_duration ,
    out object out_faithful_time_Mean, out object out_faithful_time_Prec,
    double?[] in_faithful_time , out object out_faithful_time) {
     // construct the model
     var _evidence = Variable.Bernoulli (0.5D);
     var _evidenceBlock = Variable.If (_evidence);
     var v1 = Variable.Constant(2);
     var v2 = new Range(v1);
     var faithful_size   = Variable.New<int>();
     var faithful_range  = new Range(faithful_size );
     var v3 = Variable.Constant(1D);
     var faithful_cluster_V   = Variable.DirichletSymmetric(v2, v3);
     var faithful_cluster    = SetValueRange(Variable.Array<int>(faithful_range), v2);
     using (Variable.ForEach(faithful_range )) {
         var v4 = faithful_cluster_V  ;
         faithful_cluster   [faithful_range  ] = Variable.Discrete(v4);
     }
     var v5 = v2.Clone();
     var v6 = Variable.Array<double>(v5);
     using (var v5Block = Variable.ForEach(v5)) {
         var _i_0 = v5Block.Index;
         var v7 = Variable.Constant(0D);
         var v8 = Variable.Constant(1D);
         v6[v5] = Variable.GaussianFromMeanAndPrecision(v7, v8);
     }
     var faithful_duration_Mean  = v6;
     var v9 = v2.Clone();
     var v10 = Variable.Array<double>(v9);
     using (var v9Block = Variable.ForEach(v9)) {
         var _i_0 = v9Block.Index;
         var v11 = Variable.Constant(1D);
         var v12 = Variable.Constant(1D);
         v10[v9] = Variable.Gamma(v11, v12);
     }
     var faithful_duration_Prec   = v10;
     var faithful_duration   = Variable.Array<double>(faithful_range);
     using (Variable.ForEach(faithful_range)) {
         var v13 = Variable.New<double>();
         var v15 = Variable.Copy(faithful_cluster [faithful_range ]);
         var v16 = v2.Clone();
         v15.SetValueRange(v16);
         using (Variable.Switch(v15)) {
             var v14 = faithful_duration_Mean[v15];
             v13.SetTo(v14);
         }
         var v18 = v13;
         var v19 = Variable.New<double>();
         var v21 = Variable.Copy(faithful_cluster [faithful_range ]);
         var v22 = v2.Clone();
         v21.SetValueRange(v22);
         using (Variable.Switch(v21)) {
             var v20 = faithful_duration_Prec [v21];
             v19.SetTo(v20);
         }
         var v24 = v19;
         faithful_duration  [faithful_range ] = Variable.GaussianFromMeanAndPrecision(v18, v24);
     }
     var faithful_duration_size    = Variable.New<int>();
     var faithful_duration_range   = new Range(faithful_duration_size );
     var faithful_duration_indices    = SetValueRange(Variable.Array<int>(faithful_duration_range),
            faithful_range );
     var faithful_duration_subarray    = Variable.Subarray(faithful_duration   ,
           faithful_duration_indices   );
     var v25 = v2.Clone();
     var v26 = Variable.Array<double>(v25);
     using (var v25Block = Variable.ForEach(v25)) {
         var _i_0 = v25Block.Index;
         var v27 = Variable.Constant(60D);
         var v28 = Variable.Constant(1D);
         v26[v25] = Variable.GaussianFromMeanAndPrecision(v27, v28);
     }
     var faithful_time_Mean = v26;
     var v29 = v2.Clone();
     var v30 = Variable.Array<double>(v29);
     using (var v29Block = Variable.ForEach(v29)) {
         var _i_0 = v29Block.Index;
         var v31 = Variable.Constant(1D);
         var v32 = Variable.Constant(1D);
         v30[v29] = Variable.Gamma(v31, v32);
     }
     var faithful_time_Prec  = v30;
     var faithful_time  = Variable.Array<double>(faithful_range);
     using (Variable.ForEach(faithful_range)) {
         var v33 = Variable.New<double>();
         var v35 = Variable.Copy(faithful_cluster [faithful_range ]);
         var v36 = v2.Clone();
         v35.SetValueRange(v36);
         using (Variable.Switch(v35)) {
             var v34 = faithful_time_Mean[v35];
             v33.SetTo(v34);
         }
         var v38 = v33;
         var v39 = Variable.New<double>();
         var v41 = Variable.Copy(faithful_cluster [faithful_range ]);
         var v42 = v2.Clone();
         v41.SetValueRange(v42);
         using (Variable.Switch(v41)) {
             var v40 = faithful_time_Prec [v41];
             v39.SetTo(v40);
         }
```

```
            var v44 = v39;
            faithful_time  [faithful_range ] = Variable.GaussianFromMeanAndPrecision(v38, v44);
        }
        var faithful_time_size    = Variable.New<int>();
        var faithful_time_range   = new Range(faithful_time_size );
        var faithful_time_indices    = SetValueRange(Variable.Array<int>(faithful_time_range),
            faithful_range );
        var faithful_time_subarray   = Variable.Subarray(faithful_time  , faithful_time_indices   );
        _evidenceBlock.CloseBlock();
        // observe variables using [in] parameters
        faithful_size   .ObservedValue = in_faithful_size  ;
        faithful_duration_size    .ObservedValue = mixture.ValueCount(in_faithful_duration);
        faithful_duration_indices    .ObservedValue = mixture.Indices(in_faithful_duration  );
        faithful_duration_subarray   .ObservedValue = mixture.Values(in_faithful_duration );
        faithful_time_size   .ObservedValue = mixture.ValueCount(in_faithful_time);
        faithful_time_indices    .ObservedValue = mixture.Indices(in_faithful_time  );
        faithful_time_subarray    .ObservedValue = mixture.Values(in_faithful_time );
        // infer variables to set [out] parameters
        var _inferenceEngine  = new MicrosoftResearch.Infer.InferenceEngine (_algorithm);
        _inferenceEngine .NumberOfIterations = _iterations ;
        _inferenceEngine .OptimiseForVariables  = new IVariable[] {
                _evidence ,
                faithful_cluster_V   ,
                faithful_cluster    ,
                faithful_duration_Mean  ,
                faithful_duration_Prec   ,
                faithful_duration    ,
                faithful_time_Mean ,
                faithful_time_Prec   ,
                faithful_time  };
        out_evidence  = _inferenceEngine .Infer <Bernoulli >(_evidence);
        out_faithful_cluster_V     = _inferenceEngine .Infer (faithful_cluster_V   );
        out_faithful_cluster      = _inferenceEngine .Infer (faithful_cluster   );
        out_faithful_duration_Mean    = _inferenceEngine .Infer (faithful_duration_Mean );
        out_faithful_duration_Prec    = _inferenceEngine .Infer (faithful_duration_Prec   );
        out_faithful_duration     = _inferenceEngine .Infer (faithful_duration   );
        out_faithful_time_Mean   = _inferenceEngine .Infer (faithful_time_Mean );
        out_faithful_time_Prec    = _inferenceEngine .Infer (faithful_time_Prec  );
        out_faithful_time    = _inferenceEngine .Infer (faithful_time  );
    }
}
```

D Scoping and Alpha-Equivalence

In this appendix, we give formal definitions of scoping and alpha-equivalence for Tabular, and assert some basic properties relating reduction and scoping.

Inputs, Locals and Outputs:

$\text{inputs}((c : T\ (\ell\ \textbf{input})\ \varepsilon)) = \{c\}$
$\text{inputs}((c : T\ (\ell\ \textbf{local})\ M)) = \varnothing$
$\text{inputs}((c : T\ (\ell\ \textbf{output})\ M)) = \varnothing$ otherwise

$\text{locals}((c : T\ (\ell\ \textbf{input})\ \varepsilon)) = \varnothing$
$\text{locals}((c : T\ (\ell\ \textbf{local})\ M)) = \{c\}$
$\text{locals}((c : T\ (\ell\ \textbf{output})\ M)) = \varnothing$

$\text{outputs}((c : T\ (\ell\ \textbf{input})\ \varepsilon)) = \varnothing$
$\text{outputs}((c : T\ (\ell\ \textbf{local})\ M)) = \varnothing$
$\text{outputs}((c : T\ (\ell\ \textbf{output})\ M)) = \{c\} \cup \text{outputs}(M)$ otherwise

$\text{outputs}(\varepsilon) = \{\}$
$\text{outputs}(E) = \{\}$
$\text{outputs}(\mathbb{T}\ R) = \text{outputs}(\mathbb{T}) \setminus \{\mathsf{ret}\}$
$\text{outputs}(M[e_{index} < e_{size}]) = \text{outputs}(M)$

$\text{dom}(\mathbb{T}) = \text{inputs}(\mathbb{T}) \cup \text{locals}(\mathbb{T}) \cup \text{outputs}(\mathbb{T})$.

Free Variables: $\text{fv}(R)$ **and** $\text{fv}(M)$ **and** $\text{fv}(\mathbb{T})$**:**

$\text{fv}([]) = \varnothing$
$\text{fv}((c = E) :: R) = \text{fv}(E) \cup \text{fv}(R)$

$\text{fv}(\varepsilon) = \varnothing$
$\text{fv}(E) = \text{fv}(E)$
$\text{fv}(\mathbb{T}\ R) = \text{fv}(\mathbb{T}) \cup \text{fv}(R)$
$\text{fv}(M[e_{index} < e_{size}]) = \text{fv}(M) \cup \text{fv}(e_{index}) \cup \text{fv}(e_{size})$

$\text{fv}([]) = \varnothing$
$\text{fv}(((c : T\ \ell\ viz\ M) :: \mathbb{T}) = \text{fv}(T) \cup \text{fv}(M) \cup$
$\qquad (\text{fv}(\mathbb{T}) \setminus (\{c\} \cup \text{outputs}(M)))$
$\text{fv}([]) = \varnothing$
$\text{fv}((t = \mathbb{T}) :: \mathbb{S}) = \text{fv}(\mathbb{T}) \cup \text{fv}(\mathbb{S})$

We define *alpha-equivalence* of tables as the least congruence closed under the following rule:

$$(c : T\ \ell\ \textsf{local}\ M) :: \mathbb{T} = (c' : T\ \ell\ \textsf{local}\ M) :: (\mathbb{T}\{^{c'}/_{c}\})$$
$$\text{if } c' \notin \text{outputs}(M) \cup \text{fv}(\mathbb{T})$$

In general, we identify all phrases of syntax up to alpha-equivalence, and write $\phi = \phi'$ to mean phrases ϕ and ϕ' are identical up to alpha-equivalence. See Appendix D for the detailed formal definition of alpha-equivalence.

Substitution: $\mathsf{col}\{^{E}/_{c}\}$ **and** $\mathbb{T}\{^{E}/_{c}\}$ **and** $M\{^{E}/_{c}\}$

$\mathbb{T}\{^{E}/_{c}\} = [\mathsf{col}_1\{^{E}/_{c}\}; \ldots ; \mathsf{col}_n\{^{E}/_{c}\}]$
where $\mathbb{T} = [\mathsf{col}_1; \ldots ; \mathsf{col}_n]$ and $\text{fv}(E) \cap \text{dom}(\mathbb{T}) = \varnothing$

The side-conditions define substitution as a partial function.

Lemma 2. *If* $\text{fv}(E) \cap (\text{inputs}(\mathbb{T}) \cup \text{outputs}(\mathbb{T})) = \varnothing$ *then* $\mathbb{T}\{^{E}/_{x}\}$ *is well-defined.*

We define alpha-equivalence following Andrew Pitts[31, p. 133].

Transposition for tables: $(a\ b)\mathbb{T}$

$$(a\ b)((c : T\ A\ M) :: \mathbb{T}) =
\begin{cases}
(b : (a\ b)T\ A\ (a\ b)M) :: (a\ b)\mathbb{T} & \text{if } c = a\\
(a : (a\ b)T\ A\ (a\ b)M) :: (a\ b)\mathbb{T} & \text{if } c = b\\
(c : (a\ b)T\ A\ (a\ b)M) :: (a\ b)\mathbb{T} & \text{otherwise}
\end{cases}$$

$(a\,b)[\,] = [\,]$

Transposition for models: $(a\,b)M$

$(a\,b)\varepsilon = \varepsilon$
$(a\,b)E = (a\,b)E$
$(a\,b)(\mathbb{T}\,R) = ((a\,b)\mathbb{T})\ ((a\,b)R)$
$(a\,b)(M[e_{index} < e_{size}]) = ((a\,b)M)[(a\,b)e_{index} < (a\,b)e_{size}]$

Transposition for column types: $(a\,b)T$

$(a\,b)(S\,!\,d) = (S\,!\,d)$
$(a\,b)(\mathbf{mod}(e)) = \mathbf{mod}((a\,b)e)$
$(a\,b)(T[e]) = (a\,b)T[(a\,b)e]$

Transposition for table types: $(a\,b)Q$

$$(a\,b)((c : T\ A) :: Q) = \begin{cases} (b : (a\,b)T\ A) :: (a\,b)Q & \text{if } c = a \\ (a : (a\,b)T\ A) :: (a\,b)Q & \text{if } c = b \\ (c : (a\,b)T\ A) :: (a\,b)Q & \text{otherwise} \end{cases}$$

$(a\,b)[\,] = [\,]$

Transposition for expressions: $(a\,b)E$**, interesting cases**

$$(a\,b)(\mathbf{sizeof}(t)) = \begin{cases} \mathbf{sizeof}(b) & \text{if } t = a \\ \mathbf{sizeof}(a) & \text{if } t = b \\ \mathbf{sizeof}(t) & \text{otherwise} \end{cases}$$

$$(a\,b)(E.c) = \begin{cases} ((a\,b)E).b & \text{if } c = a \\ ((a\,b)E).a & \text{if } c = b \\ ((a\,b)E).c & \text{otherwise} \end{cases}$$

Variables (Free and Bound): $\mathsf{vars}(T)$ **and** $\mathsf{vars}(Q)$

$\mathsf{vars}(S\,!\,d) = \varnothing$
$\mathsf{vars}(\mathbf{mod}(e)\,!\,d) = \mathsf{vars}(e)$
$\mathsf{vars}(T[e]) = \mathsf{vars}(T) \cup \mathsf{vars}(e)$

$\mathsf{vars}([]) = \varnothing$
$\mathsf{vars}(((c : T\ A)) :: Q) = \{c\} \cup \mathsf{vars}(T) \cup \mathsf{vars}(Q)$

Alpha equivalence for scalar types: $T_1 =_\alpha T_2$

(ALPHAEQ SCALAR)

$$\frac{}{S\,!\,d =_\alpha S\,!\,d}$$

(AlphaEq Mod)

$$\frac{e_1 =_\alpha e_2}{\mathbf{mod}(e_2) =_\alpha \mathbf{mod}(e_2)}$$

(AlphaEq Array)

$$\frac{T_1 =_\alpha T_2 \quad e_1 =_\alpha e_2}{T_1[e_2] =_\alpha T_2[e_2]}$$

Alpha equivalence for table types: $Q_1 =_\alpha Q_2$

(AlphaEq Field Type)

$$\frac{T_1 =_\alpha T_2 \quad Q_1 =_\alpha Q_2}{(c : T_1\ (\ell\ viz)) :: Q_1 =_\alpha (c : T_2\ (\ell\ viz)) :: Q_2}$$

(AlphaEq Empty Type)

$$\frac{}{[] =_\alpha []}$$

Alpha equivalence for arguments: $R_1 =_\alpha R_2$

(AlphaEq ArgExpr)

$$\frac{e_1 =_\alpha e_2 \quad R_1 =_\alpha R_2}{(c = e_1) :: R_1 =_\alpha (c = e_2) :: R_2}$$

(AlphaEq ArgEmpty)

$$\frac{}{[] =_\alpha []}$$

Alpha equivalence for model expressions: $M_1 =_\alpha M_2$

(AlphaEq ModelExpr)

$$\frac{E_1 =_\alpha E_2}{E_1 =_\alpha E_2}$$

(AlphaEq ModelAppl)

$$\frac{\mathbb{T}_1 =_\alpha \mathbb{T}_2 \quad R_1 =_\alpha R_2}{\mathbb{T}_1\ R_1 =_\alpha \mathbb{T}_2\ R_2}$$

(AlphaEq ModelIndexed)

$$\frac{M_1 =_\alpha M_2 \quad E_1 =_\alpha E_2 \quad e_3 =_\alpha e_3}{M_1[E_1 < e_3] =_\alpha M_2[E_2 < e_4]}$$

Alpha equivalence for tables: $\mathbb{T}_1 =_\alpha \mathbb{T}_2$

(AlphaEq Local)

$$\frac{T_1 =_\alpha T_2 \quad M_1 =_\alpha M_2 \quad (c_1\ c)\mathbb{T}_1 =_\alpha (c_2\ c)\mathbb{T}_2 \\ c \notin \mathsf{vars}(c_1, c_2, \mathbb{T}_1, \mathbb{T}_2)}{(c_1 : T_1\ (\ell\ \mathbf{local})\ M_1) :: \mathbb{T}_1 =_\alpha (c_2 : T_2\ (\ell\ \mathbf{local})\ M_2) :: \mathbb{T}_2}$$

(AlphaEq Field)

$$\frac{T_1 =_\alpha T_2 \quad M_1 =_\alpha M_2 \quad \mathbb{T}_1 =_\alpha \mathbb{T}_2 \quad viz \neq \textsf{local}}{(c : T_1\ (\ell\ viz)\ M_1) :: \mathbb{T}_1 =_\alpha (c : T_2\ (\ell\ viz)\ M_2) :: \mathbb{T}_2}$$

(AlphaEq Empty)

$$\frac{}{[] =_\alpha []}$$

Properties of the Reduction System Reduction preserves the exports of a table:

Lemma 3. (1) *If* $\mathbb{S} \to \mathbb{S}'$ *then* $\mathrm{outputs}(\mathbb{S}) = \mathrm{outputs}(\mathbb{S}')$.
(2) *If* $\mathbb{T} \to \mathbb{T}'$ *then* $\mathrm{outputs}(\mathbb{T}) = \mathrm{outputs}(\mathbb{T}')$.
(3) *If* $M \to M'$ *then* $\mathrm{outputs}(M) = \mathrm{outputs}(M')$.

Lemma 4. *If* $\mathbb{T} \to \mathbb{T}'$ *then* $\mathrm{inputs}(\mathbb{T}) = \mathrm{inputs}(\mathbb{T}')$.

As usual, reduction cannot introduce new free variables:

Lemma 5. (1) *If* $\mathbb{S} \to \mathbb{S}'$ *then* $\mathrm{fv}(\mathbb{S}') \subseteq \mathrm{fv}(\mathbb{S})$.
(2) *If* $\mathbb{T} \to \mathbb{T}'$ *then* $\mathrm{fv}(\mathbb{T}') \subseteq \mathrm{fv}(\mathbb{T})$.
(3) *If* $M \to M'$ *then* $\mathrm{fv}(M') \subseteq \mathrm{fv}(M)$.

Lemma 6. (1) *If* $\mathbb{S} \to \mathbb{S}'$ *and* $\mathbb{S} \to \mathbb{S}''$ *then* $\mathbb{S} =_\alpha \mathbb{S}''$.
(2) *If* $\mathbb{T} \to \mathbb{T}'$ *and* $\mathbb{T} \to \mathbb{T}''$ *then* $\mathbb{T} =_\alpha \mathbb{T}''$.
(3) *If* $M \to M'$ *and* $M \to M''$ *then* $M =_\alpha M''$.

Proof: By inspection of the inference rules. ■

E Semantics of Core Tabular

In this appendix, we present in detail the denotational semantics for probabilistic attributes of Tabular, as well as a set of rules for evaluating **qry** attributes and computing the output database corresponding to a given program and its input values.

E.1 Measure theory semantics

Monadic Semantics of Tabular schemas Following the work of pioneers including Kozen [20], Giry [12], and Ramsey and Pfeffer [32], we base the semantics of our probabilistic language on measure theory.

The standard definition of a *measurable space* is a pair (Ω, Σ) where Ω is a set of possible outcomes, and $\Sigma \subseteq \mathscr{P}(\Omega)$ is a σ-algebra, that is, Σ is a set that (1) contains $\varnothing$ and Ω, and (2) is closed under complement and countable union and intersection.

- Let $* \triangleq (\{()\}, \{\{()\}, \varnothing\})$.
- We use the notation $\sigma_\Omega(S)$, when $S \subseteq \mathscr{P}(\Omega)$, for the least σ-algebra over Ω that is a superset of S.

- Given two measurable spaces (Ω_1, Σ_1) and (Ω_2, Σ_2), we can compute their product as:
$$(\Omega_1, \Sigma_1) \times (\Omega_2, \Sigma_2) \triangleq (\Omega_1 \times \Omega_2, \sigma_{\Omega_1 \times \Omega_2}\{A \times B \mid A \in \Sigma_1, B \in \Sigma_2\})$$
- Define $(\Omega, \Sigma)^0 \triangleq *$.
- For $n > 0$, define $(\Omega, \Sigma)^n$ by $(\Omega, \Sigma)^{n+1} \triangleq (\Omega, \Sigma) \times (\Omega, \Sigma)^n$.

For each closed type T we define a measurable space $\boldsymbol{T}[\![T]\!]$ as follows. The set $\sigma_{\mathbb{R}}(\{[a,b] \mid a,b \in \mathbb{R}\})$ in the definition of $\boldsymbol{T}[\![\textbf{real}]\!]$ is the Borel σ-algebra on the real line, which is the smallest σ-algebra containing all closed (and open) intervals. Given two measurable spaces (Ω_1, Σ_1) and (Ω_2, Σ_2), there is a product, the measurable space $(\Omega_1, \Sigma_1) \times (\Omega_2, \Sigma_2)$, whose set of outcomes consists of pairs (ω_1, ω_2) where $\omega_i \in \Omega_i$. Hence, the notation $\boldsymbol{T}[\![T]\!]^n$ represents an array type by the n-fold product of $\boldsymbol{T}[\![T]\!]$.

The measurable space of a table is defined recursively as a product of the first column of the table (which is itself a product space in case of **inst**-level columns) and the rest of the table, or the trivial space $*$ in case of an empty table. Similarly, the measurable space of a schema is the product of the space of the first table and the rest of the schema, or $*$ if the schema is empty.

In the probabilistic and query semantics for Tabular, we assume an input database $DB = (\delta_{in}, \rho_{sz})$, where ρ_{sz} is a map containing the table sizes, and δ_{in} is a map storing the values of input columns (indexed by table and column names).

Closed Types, Tables, and Schemas as Measurable Spaces: $\boldsymbol{T}[\![T]\!]_{\rho_{sz}}$, $\boldsymbol{T}[\![\mathbb{T}]\!]_t^{\rho_{sz}}$, $\boldsymbol{T}[\![\mathbb{S}]\!]^{\rho_{sz}}$

$\boldsymbol{T}[\![\textbf{bool}!spc]\!](\mathbb{B}, \mathscr{P}(\mathbb{B}))$
$\boldsymbol{T}[\![\textbf{int}!spc]\!]_{\rho_{sz}} = (\mathbb{Z}, \mathscr{P}(\mathbb{Z}))$
$\boldsymbol{T}[\![\textbf{real}!spc]\!]_{\rho_{sz}} = (\mathbb{R}, \sigma_{\mathbb{R}}(\{[a,b] \mid a,b \in \mathbb{R}\}))$
$\boldsymbol{T}[\![\textbf{mod}(n)!spc]\!]_{\rho_{sz}} = (\{0\}, \{\{0\}, \varnothing\})$ where $n \leq 0$
$\boldsymbol{T}[\![\textbf{mod}(n)!spc]\!]_{\rho_{sz}} = (0..n-1, \mathscr{P}(0..n-1))$ where $n > 0$
$\boldsymbol{T}[\![\textbf{mod}(\textbf{sizeof}(t))!spc]\!]_{\rho_{sz}} = (0..\rho_{sz}(t)-1, \mathscr{P}(0..\rho_{sz}(t)-1))$

$\boldsymbol{T}[\![T[n]]\!]_{\rho_{sz}} = *$ where $n \leq 0$
$\boldsymbol{T}[\![T[n]]\!]_{\rho_{sz}} = \boldsymbol{T}[\![T]\!]^n_{\rho_{sz}}$ where $n > 0$
$\boldsymbol{T}[\![T[\textbf{sizeof}(t)]]\!]_{\rho_{sz}} = \boldsymbol{T}[\![T]\!]^{\rho_{sz}(t)}_{\rho_{sz}}$

$\boldsymbol{T}[\![[]]\!]_t^{\rho_{sz}} \triangleq *$
$\boldsymbol{T}[\![(c : T\ \ell\ \textbf{input}\ \varepsilon) :: \mathbb{T}]\!]_t^{\rho_{sz}} \triangleq \boldsymbol{T}[\![\mathbb{T}]\!]_t^{\rho_{sz}}$
$\boldsymbol{T}[\![(c : T\ \ell\ viz\ E) :: \mathbb{T}]\!]_t^{\rho_{sz}} \triangleq \boldsymbol{T}[\![\mathbb{T}]\!]_t^{\rho_{sz}}$ if $\text{space}(T) = \textbf{qry}$
$\boldsymbol{T}[\![(c : T\ \textbf{static}\ viz\ E) :: \mathbb{T}]\!]_t^{\rho_{sz}} \triangleq (\boldsymbol{T}[\![T]\!]_{\rho_{sz}}) \times (\boldsymbol{T}[\![\mathbb{T}]\!]_t^{\rho_{sz}})$ if $\text{space}(T) \neq \textbf{qry}$
$\boldsymbol{T}[\![(c : T\ \textbf{inst}\ viz\ E) :: \mathbb{T}]\!]_t^{\rho_{sz}} \triangleq (\boldsymbol{T}[\![T]\!]^{\rho_{sz}(t)}_{\rho_{sz}}) \times (\boldsymbol{T}[\![\mathbb{T}]\!]_t^{\rho_{sz}})$ if $\text{space}(T) \neq \textbf{qry}$

$\boldsymbol{T}[\![[]]\!]^{\rho_{sz}} \triangleq *$
$\boldsymbol{T}[\![(t = \mathbb{T}) :: \mathbb{S}]\!]^{\rho_{sz}} \triangleq (\boldsymbol{T}[\![\mathbb{T}]\!]_t^{\rho_{sz}}) \times (\boldsymbol{T}[\![\mathbb{S}]\!]^{\rho_{sz}})$

Lemma 7. *Consider a closed type T. If $\mathbf{T}[\![T]\!] = (\Omega, \Sigma)$ then $\Omega = \{V \mid \varnothing \vdash^{pc} V : T\}$.*

Proof: The proof is by induction on the size of T. ■

Next, we recall standard definitions of measure and integration. (See, for example, Panangaden [28] for a readable introduction.)

- A *measure* μ on a measurable space (Ω,Σ) is a function $\Sigma \to \mathbb{R}^+ \cup \{\infty\}$ that is countably additive, that is, $\mu(\varnothing) = 0$ and if the sets $A_0, A_1, \ldots \in \Sigma$ are pairwise disjoint, then $\mu(\cup_i A_i) = \sum_i \mu(A_i)$. We write $|\mu| \triangleq \mu(\Omega)$. It is called a probability measure if $|\mu| = 1$, and a sub-probability measure if $|\mu| \leq 1$. If $r \geq 0$, we write $r \cdot \mu$ for the measure such that $(r \cdot \mu)(A) = r(\mu(A))$.
- If μ is a measure on (Ω,Σ), and f is a non-negative (measurable) function $\Sigma \to \mathbb{R}$, we write $\int f(x)\, d\mu(x)$ for Lebesgue integration of f with respect to μ.
- The (independent) product $(\mu_1 \times \mu_2)$ of two measures is also definable [2, Sec. 18], and satisfies $(\mu_1 \times \mu_2)(A \times B) = \mu_1(A) \cdot \mu_2(B)$.
- Hence, we can define an (independent) product $\texttt{sequence}[\mu_1, \ldots, \mu_n]$ of n measures such that $\texttt{sequence}[\mu_1, \ldots, \mu_n]\ (A_1 \times \cdots \times A_n) = \mu_1(A_1)\texttt{sequence}(\mu_2, \ldots, \mu_n)$ for $n > 0$ and $\texttt{sequence([])} = *$.

Recall that a *tagged value* a is either $\textsf{static}(V)$ or $\textsf{inst}(V)$ where V is an array of values. The measure theory semantics makes use of the following data structures:

Data structures used in the probabilistic semantics:

$\delta ::= [t_i \mapsto \tau_i{}^{i \in 1..n}]$	schema-level environment

For continuous observations, we need a relation stating when two values are approximately (that is, up to some finite precision) equal. The expression on the left-hand side is always taken from a database, so it can be either empty or a tagged value.

Rules of Matching: $a \preceq V'$ **where** known(V')

(MATCH ?)
$$\frac{}{\ell(?) \preceq V}$$

(MATCH FLOAT)
$$\frac{r = \mathsf{round}(s)}{\ell(r) \preceq s}$$

(MATCH SCALAR)
$$\frac{\mathsf{ty}(s) \in \{\textsf{bool}, \textsf{int}\}}{\ell(s) \preceq s}$$

(MATCH ARRAY)
$$\frac{\ell(V_i) \preceq V_i' \quad \forall i \in 0..n-1}{\ell([V_0; \ldots; V_{n-1}]) \preceq [V_0'; \ldots; V_{n-1}']}$$

We define the operators of the probability monad, an operation for conditioning by filtering, and a function `sequence`, which takes an array of probability distributions, and yields a probability distribution over arrays, defined as the product distribution of the distributions in the array.

Probability Monad and Conditioning:

$(\mu \gg= f)\ A \triangleq \int f(x)(A)\, d\mu(x)$	monadic bind
$(\texttt{return}\, V)\ A \triangleq 1$ if $V \in A$, else 0	monadic return
$([a \preceq V]\mu)A \triangleq \mu(A)$ if $(a \preceq V)$, else 0	conditioning by filtering

Lemma 8. $(\texttt{return}\, V \gg\!= f) = f(V)$.

Proof: Let $g_S(x) = f(x)(S)$. Since for every S, g_S is a measurable function, there exists an increasing sequence of measurable simple functions (i.e. admitting only a finite number of values) $\{g_{S,m}\}$ which coverges pointwise to f, i.e. $g_S(x) = \lim_{m\to\infty} g_{S,m}(x)$ for all x. Thus, for every S, $\int g_S \,\mathrm{d}\mu = \lim_{m\to\infty} \int g_{S,m} \mathrm{d}\mu$.

Each simple function h can be represented as $\sum_{i=1}^{n} \alpha_i \mathbb{1}_{A_i}$, where $\alpha_1, \ldots, \alpha_n$ are the values taken by h, $A_i = h^{-1}(\alpha_i)$ for all i and $\mathbb{1}_A(x)$ is then standard indicator function, returning 1 if $x \in A$ and 0 otherwise. The Lebesgue integral for h then is defined as $\int h \,\mathrm{d}\mu = \sum_{i=1}^{n} \alpha_i \mu(A_i)$.

Now, let $g_{S,m} = \sum_{i=1}^{n_{S,m}} \alpha_{S,m,i} \mathbb{1}_{A_{S,m,i}}$ for every S, m. Then, $\int g_{S,m} \,\mathrm{d}\mu = \sum_{i=1}^{n_{S,m}} \alpha_{S,m,i} \mu(A_{S,m,i})$.

Thus, for $\mu = \texttt{return}\, V$ (i.e. $\mu(A) = \mathbb{1}_A(V)$), we have:

$$\begin{aligned}
\texttt{return}\, V \gg\!= f &= \lambda S. \int f(x)(S) \,\mathrm{d}\mu(x) \\
&= \lambda S. \int g_S(x) \,\mathrm{d}\mu(x) \\
&= \lambda S. \int g_S \,\mathrm{d}\mu \\
&= \lambda S. \lim_{m\to\infty} \int g_{S,m} \,\mathrm{d}\mu \\
&= \lambda S. \lim_{m\to\infty} \sum_{i=1}^{n_{S,m}} \alpha_{S,m,i} \mu(A_{S,m,i}) \\
&= \lambda S. \lim_{m\to\infty} \sum_{i=1}^{n_{S,m}} \alpha_{S,m,i} \mathbb{1}_{A_{S,m,i}}(V) \\
&= \lambda S. \lim_{m\to\infty} g_{S,m}(V) \\
&= \lambda S. g_S(V) \\
&= \lambda S. f(V)(S) \\
&= f(V)
\end{aligned}$$

as required.

■

Lemma 9. *If* $\varnothing \vdash^{pc} V : T$ *then* $\texttt{return}\, V$ *is a probability measure on* $\mathbf{T}[\![T]\!]_{\rho_{sz}}$.

Proof: First, we need to show that $\texttt{return}\, V$ is indeed a measure. For this, we require the following properties:

- $(\texttt{return}\, V)\,\varnothing = 0$
  Trivial- there is no V such that $V \in \varnothing$.
- If $A_1, A_2, \ldots \in \Sigma$ are pairwise disjoint, then $(\texttt{return}\, V)(\bigcup A_i) = \sum_i (\texttt{return}\, V)\,(A_i)$.
  Since the sets $A_1, A_2, \ldots$ are pairwise disjoint, V cannot belong to more than one set. Thus, $\sum_i (\texttt{return}\, V)\,(A_i) = 1$ if $V \in A_i$ for some i, 0 otherwise. As $V \in A_i$ for some i if and only if $V \in \bigcup A_i$, the required equality holds.

Finally, we have to show that the measure `return` V is a probability measure, i.e. that $|\mu| = \mu(\Omega) = 1$, where $\boldsymbol{T}[\![T\,!spc]\!] = (\Omega, \Sigma)$.

We have $\mu(\Omega) = 1$ if $V \in \Omega$, else 0. By lemma 7, $\Omega = \{V' | \varnothing \vdash V' : T\}$. Hence, by the assumption $\varnothing \vdash V : T$, we have $V \in \Omega$, which implies $\mu(\Omega) = 1$, as required.

Therefore, `return` V is a probability measure on $\boldsymbol{T}[\![T]\!]$. ■

Lemma 10. *If μ is a probability measure on $\mathbf{T}[\![T]\!]_{\rho_{sz}}$ and $f(V)$ is a probability measure on $\mathbf{T}[\![U]\!]_{\rho_{sz}}$ whenever $\varnothing \vdash^{pc} V : T$, then $\mu \gg\!= f$ is a probability measure on $\mathbf{T}[\![U]\!]_{\rho_{sz}}$.*

Proof: Let $\boldsymbol{T}[\![T]\!] = (\Omega_T, \Sigma_T)$ and $\boldsymbol{T}[\![U]\!] = (\Omega_U, \Sigma_U)$.

As before we need to show that the three properties of probablilty measures hold:

- $(\mu \gg\!= f)(\varnothing) = 0$
  By lemma 7, $\Omega_T = \{V | \varnothing \vdash V : T\}$, which implies that $f(x)$ is a measure for all $x \in \Omega_T$, and so $f(x)(\varnothing)$ for all x. By a property of Lebesgue integral, $\int f(x)(\varnothing)\, d\mu(x) = 0$, as required
- If $A_1, A_2, \ldots \in \Sigma$ are pairwise disjoint, then $(\mu \gg\!= f)(\bigcup A_i) = \sum_i (\mu \gg\!= f)\,(A_i)$ (i.e. $\int f(x)(\bigcup A_i)\, d\mu(x) = \sum_i \int f(x)(A_i)\, d\mu(x)$).
  Since $f(x)$ is a measure for all $x \in \Omega_T$, we have $f(x)(\bigcup A_i) = \sum_i f(x)(A_i)$ by definition of a measure. Then, the countable additivity property of the Lebesgue integral yields $\int \sum_i f(x)(A_i)\, d\mu(x) = \sum_i \int f(x)(A_i)\, d\mu(x)$. This gives $(\mu \gg\!= f)(\bigcup A_i) = \sum_i (\mu \gg\!= f)\,(A_i)$, as required.
- $|(\mu \gg\!= f)| = 1$, i.e. $(\mu \gg\!= f)(\Omega_U) = 1$
  We have $(\mu \gg\!= f)(\Omega_U) = \int f(x)(\Omega_U)\, d\mu(x)$. Since $f(x)$ is a measure on (Ω_U, Σ_U) for every $x \in \Omega_T$, we have $f(x)(\Omega_U) = 1$ for all $x \in \Omega_T$. Thus, $f(x)(\Omega_U)$ is a simple function of x which can be written as $1 * \mathbb{1}_{\Omega_T}$, and by definition of the Lebesgue integral for a simple funciton, $\int 1 * \mathbb{1}_{\Omega_T}\, \mathrm{d}\mu = \mu(\Omega_T)$. Since μ is a probability measure on (Ω_U, Σ_U), we have $\mu(\Omega_T) = 1$, as required.

■

E.2 Well-definedness of measures

In the following definition, $\rho_{sz}(T)$ denotes T with each expression of the form **sizeof**(t) replaced with $\rho_{sz}(t)$.

Size Substitution in Types and Column Type Expansion

$[\![T]\!]^{\mathsf{static},t}_{\rho_{sz}} \triangleq \rho_{sz}(T)$
$[\![T]\!]^{\mathsf{inst},t}_{\rho_{sz}} \triangleq \rho_{sz}(T)[\rho_{sz}(t)]$

In order to specify the desired properties of the semantics of schemas, we need to define a conformance relation, which states that a given database or compound evaluation environment is valid with respect to the given schema type or typing environment.

The judgment $\Gamma \vdash (\delta, \rho_{sz})$ means that the names of all the tables in Γ are present in the map of table sizes ρ_{sz} and in the schema-level evaluation environment δ, and that the maps stored in δ under these identifiers conform to the table types in Γ. The judgment $\Gamma \vdash (\delta, \tau, \rho_{sz}, t)$, in addition to the above, states that all other variables in Γ must occur in τ, and the values stored for them in τ must inhabit their types in Γ (types are expanded to array types for **inst**-level variables).

The judgment $t \mapsto \tau \models^{io}_{\rho_{sz}} Q$ means that the values in the valuation environment (which may be a part of a database) check against the types in Q, while $(\delta, \rho_{sz}) \models^{io} Sty$ states that all tables in Sty have corresponding entries in the database δ_{in}.

Conformance Relations:

$io ::= \textsf{in} \mid \textsf{out}$	polarity
$(\delta, \rho_{sz}) \models^{io} Sty$	database (δ, ρ_{sz}) conforms to schema type Sty
$t \mapsto \tau \models^{io}_{\rho_{sz}} Q$	io-table W size $\rho_{sz}(t)$ conforms to table type Q
$\Gamma \vdash (\delta, \rho_{sz})$	environment Γ makes sense for (δ, ρ_{sz})
$\Gamma \vdash (\delta, \tau, \rho_{sz}, t)$	environment Γ makes sense for $(\delta, \tau, \rho_{sz}, t)$

Conformance Rules:

(CONF SCHEMA)

$$\frac{t_i \mapsto \tau_i \models^{io}_{\rho_{sz}} Q_i \quad \forall i \in 1..n}{([t_i \mapsto \tau_i{}^{\,i \in 1..n}], \rho_{sz}) \models^{io} [(t_i : Q_i)^{\,i \in 1..n}]}$$

(CONF TABLE IN)

$$\frac{\begin{array}{l} I = \{ j \in 1..m \mid (viz_j = \textsf{input}) \vee (viz_j = \textsf{output} \wedge \textsf{rnd}(T_j)) \} \\ \varnothing \vdash^{\ell_j} V_j : [\![T_j]\!]^{\ell_j, t}_{\rho_{sz}} \quad viz_j = \textsf{input} \Rightarrow \text{known}(V_j) \quad \forall j \in I \end{array}}{t \mapsto [c_j \mapsto \ell_j(V_j)^{\,j \in I}] \models^{\textsf{in}}_{\rho_{sz}} [(c_j : T_j\ \ell_j\ viz_j)^{\,j \in 1..m}]}$$

(CONF TABLE OUT)

$$\frac{\begin{array}{l} O = \{ j \in 1..m \mid spc(T_j) \neq \textsf{rnd} \} \\ \varnothing \vdash^{\ell_j} V_j : [\![T_j]\!]^{\ell_j, t}_{\rho_{sz}} \quad \text{known}(V_j) \quad \forall j \in O \end{array}}{t \mapsto [c_j \mapsto \ell_j(V_j)^{\,j \in O}] \models^{\textsf{out}}_{\rho_{sz}} [(c_j : T_j\ \ell_j\ viz_j)^{\,j \in 1..m}]}$$

(CONF TABLE ALL)

$$\frac{\begin{array}{l} O = \{ j \in 1..m \mid spc(T_j) \neq \textsf{qry} \} \\ \tau(c_j) = \ell_j(V_j) \quad \varnothing \vdash^{\ell_j} V_j : [\![T_j]\!]^{\ell_j, t}_{\rho_{sz}} \quad \forall j \in O \end{array}}{t \mapsto \tau \models^{\textsf{all}}_{\rho_{sz}} [(c_j : T_j\ \ell_j\ viz_j)^{\,j \in 1..m}]}$$

(VAL EMPTY SCHEMA)

$$\frac{}{\varnothing \vdash (\delta, \rho_{sz})}$$

(VAL TABLE SCHEMA)

$$\frac{(t \mapsto \delta(t)) \models^{\textsf{all}}_{\rho_{sz}} Q \quad t \in \text{dom}(\rho_{sz}) \quad \Gamma \vdash (\delta, \rho_{sz})}{\Gamma, t : Q \vdash (\delta, \rho_{sz})}$$

(VAL EMPTY TABLE-EXPR)

$$\frac{}{\varnothing \vdash (\delta, \tau, \rho_{sz}, t)}$$

(VAL TABLE TABLE-EXPR)

$$\frac{(t \mapsto \delta(t)) \models^{\mathsf{all}}_{\rho_{sz}} Q \quad t \in \mathrm{dom}(\rho_{sz}) \quad \Gamma \vdash (\delta, \rho_{sz})}{\Gamma, t : Q \vdash (\delta, \tau, \rho_{sz}, t)}$$

(VAL VAR TABLE-EXPR)

$$\frac{\tau(c) = \ell(V) \quad \varnothing \vdash^{\ell} V : [\![T]\!]^{\ell,t}_{\rho_{sz}} \quad \Gamma \vdash (\delta, \tau, \rho_{sz}, t)}{\Gamma, c :^{\ell} T \vdash (\delta, \tau, \rho_{sz}, t)}$$

The semantics of an expression E is a probability measure $\boldsymbol{P}[\![E]\!]^{i}_{(\tau,\delta)}$, where i is the index of the row of the table in which the expression resides. This index is set to 0, and not taken into account, for expressions in **static** columns. The semantics of a table $\mathbb{T}$ with name t is a sub-probability measure $\boldsymbol{P}[\![\mathbb{T}]\!]^{t,\tau}_{(\tau,\delta)}$, where τ is the part of the input database corresponding to this table, δ a map storing the values of non-**qry** columns of all previous tables and τ the current table-level environment (initially equal to $[]$). Finally, the semantics of a schema $\mathbb{S}$ is a sub-probability measure $\boldsymbol{P}[\![\mathbb{S}]\!]^{\delta_{in}}_{\delta}$ where δ adds values to the schema-level valuation environment in the recursive calls (and is equal to $[]$ initially).

Given a closed well-typed primitive $D[n_1, \dots, n_m](V_1, \dots, V_n)$ of closed type T, we assume a finite measure $\mu_{D[n_1,\dots,n_m](V_1,\dots,V_n)}$ on measurable space $\boldsymbol{T}[\![T]\!]$. We also assume a global map ρ_{sz} storing sizes of tables in the database.

Semantics of Expression E as a Measure $\boldsymbol{P}[\![E]\!]^{i}_{(\tau,\delta)}$ on $\boldsymbol{T}[\![T]\!]$:

We assume that $z, z_1, \dots, z_n$ do not occur free in $E, F, F_1, E_1, \dots, E_n, x, \tau$.

$\mathsf{eval}^{i}_{\rho_{sz}}(s) = s$ Evaluation of index expression
$\mathsf{eval}^{i}_{\rho_{sz}}(x) =$
  $\textbf{match } \tau(x) \textbf{ with static}(V_1) \to V_1 \mid \textbf{inst}(V_2) \to V_2[i]$
$\mathsf{eval}^{i}_{\rho_{sz}}(\textbf{sizeof}(t)) = \rho_{sz}(t)$

$\boldsymbol{P}[\![E]\!]^{i}_{(\tau,\delta)} \triangleq \boldsymbol{P}[\![E]\!]^{i}$ **where**
$\boldsymbol{P}[\![e]\!]^{i} \triangleq \texttt{return}\ \mathsf{eval}^{i}_{\tau,\rho_{sz}}(e)$
$\boldsymbol{P}[\![[E_0; \dots; E_{n-1}]]\!]^{i} \triangleq \boldsymbol{P}[\![E_0, \dots, E_{n-1}]\!]^{i}(z_1, \dots, z_n)[\texttt{return}\ [z_1; \dots; z_n]]$
$\boldsymbol{P}[\![E[F]]\!]^{i} \triangleq \boldsymbol{P}[\![E]\!]^{i} \gg= \lambda z.\boldsymbol{P}[\![F]\!]^{i} \gg= \lambda w.\texttt{return}\ z[w]$
$\boldsymbol{P}[\![\textbf{if } E \textbf{ then } F_1 \textbf{ else } F_2]\!]^{i} \triangleq \boldsymbol{P}[\![E]\!]^{i} \gg= \lambda z.\texttt{if}\ z\ \texttt{then}\ \boldsymbol{P}[\![F_1]\!]^{i}\ \texttt{else}\ \boldsymbol{P}[\![F_2]\!]^{i}$

$\boldsymbol{P}[\![[\textbf{for } x < e \to F]]\!]^{i} \triangleq$
  $\texttt{sequence}\ (\texttt{map}\ (\lambda v.\boldsymbol{P}[\![F]\!]^{i}_{(\tau@[x \mapsto \textbf{static } v],\delta)})\ (0..(n-1)))$ if $\mathsf{eval}^{i}_{\rho_{sz}}(e) = n$

$\boldsymbol{P}[\![g(E_1, \dots, E_n)]\!]^{i} \triangleq \boldsymbol{P}[\![E_1, \dots, E_n]\!]^{i}(z_1, \dots, z_n)[\texttt{return}\ \underline{g}(z_1, \dots, z_n)]$
$\boldsymbol{P}[\![D[e_1, \dots, e_m](F_1, \dots, F_n)]\!]^{i} \triangleq$
  $\boldsymbol{P}[\![F_1, \dots, F_n]\!]^{i}(z_1, \dots, z_n)[\mu_{D[n_1,\dots,n_m](z_1,\dots,z_n)}]$ for $\mathsf{eval}^{i}_{\rho_{sz}}(e_j) = n_j$

$\boldsymbol{P}[\![E_1, \dots, E_n]\!]^{i}(z_1, \dots, z_n)[\cdot] \triangleq$
  $\boldsymbol{P}[\![E_1]\!]^{i} \gg= \lambda z_1.\boldsymbol{P}[\![E_2]\!]^{i} \gg= \dots \gg= \lambda z_{n-1}.\boldsymbol{P}[\![E_n]\!]^{i} \gg= \lambda z_n.[\cdot]$

$\boldsymbol{P}[\![t.c]\!]^{i} \triangleq \texttt{return}\ \textbf{match } \delta(t).c \textbf{ with static}(V) \to V$

$$\boldsymbol{P}[\![E : t.c]\!]^i \triangleq \boldsymbol{P}[\![E]\!]^i \gg= \\ \quad \lambda z.\texttt{return}\ \textbf{match}\ \delta(t).c\ \textbf{with}\ \textbf{inst}([V_0,\dots,V_{n-1}]) \to V_z \qquad \text{if } \rho_{sz}(t) = n$$

Lemma 11. *If $\Gamma \vdash (\delta,\tau,\rho_{sz},t)$ and $\Gamma \vdash^{pc} E_i : T_i$ and $f(V_1,\dots,V_n)$ is a probability measure on $\mathbf{T}[\![U]\!]_{\rho_{sz}}$ when $\varnothing \vdash^{pc} V_i : \rho_{sz}(T_i)$ for each $i \in 1..n$,then*

(1) *If $pc =$ **static** and $\mathbf{P}[\![E_i]\!]^0_{(\tau,\delta)}$ is a probability measure on $\mathbf{T}[\![T_i]\!]_{\rho_{sz}}$ for each $i \in 1..n$, then $\mathbf{P}[\![E_1,\dots,E_n]\!]^0_{(\tau,\delta)}(z_1,\dots,z_n)[f(z_1,\dots,z_n)]$ is a probability measure on $\mathbf{T}[\![U]\!]_{\rho_{sz}}$.*

(2) *If $pc =$ **inst** and $j < \rho_{sz}(t)$ and $\mathbf{P}[\![E_i]\!]^j_{(\tau,\delta)}$ is a probability measure on $\mathbf{T}[\![T_i]\!]_{\rho_{sz}}$ for each $i \in 1..n$ and $0 \le i \le \rho_{sz}(t)-1$, then $\mathbf{P}[\![E_1,\dots,E_n]\!]^i_{(\tau,\delta)}(z_1,\dots,z_n)[f(z_1,\dots,z_n)]$ is a probability measure on $\mathbf{T}[\![U]\!]_{\rho_{sz}}$.*

Proof: By induction on n. For clarity, let us invert the list of arguments of f and write $\boldsymbol{P}[\![E_n,\dots,E_1]\!]^i_{(\tau,\delta)}(z_1,\dots,z_n)[f(z_n,\dots,z_1)]$.

- Case $n = 1$: $\boldsymbol{P}[\![E_1]\!]^i_{(\tau,\delta)}(z_1,\dots,z_n)[f(z_1)]$
  By assumption, $\boldsymbol{P}[\![E_1]\!]^i_{(\tau,\delta)}$ is a probability measure on $\tau(T)$, and for all V such that $\varnothing \vdash V : \tau(T)$, $f(V)$ us a probability measure on $\boldsymbol{T}[\![U]\!]$, so by lemma 10, $\boldsymbol{P}[\![E_1]\!]^i_{(\tau,\delta)}(z_1,\dots,z_n)[f(z_1)]$ is a probability measure on $\boldsymbol{T}[\![U]\!]$.
- Case $n > 1$: $\boldsymbol{P}[\![E_n,\dots,E_1]\!]^i_{(\tau,\delta)}(z_1,\dots,z_n)[f(z_n,\dots,z_1)]$
  We have $\boldsymbol{P}[\![E_n,\dots,E_1]\!]^i_{(\tau,\delta)}(z_1,\dots,z_n)[f(z_n,\dots,z_1)] =$
  $\boldsymbol{P}[\![E_n]\!]^i_{(\tau,\delta)} \gg= \lambda z_n.\boldsymbol{P}[\![E_{n-1}]\!]^i_{(\tau,\delta)} \gg= \dots \gg= \lambda z_1.f(z_n,\dots,z_1)$, where $\boldsymbol{P}[\![E_i]\!]^i_{(\tau,\delta)}$ is a probability measure on $\boldsymbol{T}[\![\tau(T_i)]\!]$ for all i.
  Let $g(x) = \boldsymbol{P}[\![E_{n-1},\dots,E_1]\!]^i_{(\tau,\delta)}(z_1,\dots,z_n)[f(x,z_{n-1},\dots,z_1)] =$
  $\boldsymbol{P}[\![E_{n-1}]\!]^i_{(\tau,\delta)} \gg= \dots \gg= \lambda z_1.f(x,z_{n-1},\dots,z_1)$. Now, take an arbitrary V_n such that $\varnothing \vdash V_n : \tau(T_n)$ and let $h_{V_n}(z_{n-1},\dots,z_1) = f(V_n,z_{n-1},\dots,z_1)$. By assumption, for all $V_{n-1},\dots,V_1$ such that $\varnothing \vdash V_i : \tau(T_i)$, $h_{V_n}(V_{n-1},\dots,V_1)$ is a probability measure on $\boldsymbol{T}[\![U]\!]$. Thus, by induction hypothesis,
  $g(V_n) = \boldsymbol{P}[\![E_{n-1},\dots,E_1]\!]^i_{(\tau,\delta)}(z_1,\dots,z_n)[h_{V_n}(z_{n-1},\dots,z_1)]$ is a probability measure on $\boldsymbol{T}[\![U]\!]$.
  Therefore, by lemma 10, $\boldsymbol{P}[\![E_n,\dots,E_1]\!]^i_{(\tau,\delta)}(z_1,\dots,z_n)[f(z_n,\dots,z_1)] =$
  $\boldsymbol{P}[\![E_n]\!]^i_{(\tau,\delta)} \gg= g$ is a probability measure on $\boldsymbol{T}[\![U]\!]$.

■

In the following, we expect that the expression E is used in a table t: we have that the expression E is well-typed in environment Γ, and the judgment $\Gamma \vdash (\delta,\tau,\rho_{sz},t)$ says that the table types in Γ type the tables in δ, and the attribute types in Γ type the earlier attributes in τ, with **inst**-level attributes of size $\rho_{sz}(t)$.

Lemma 12. *If $\Gamma \vdash^{pc} E : T$ and T is in **det** or **rnd**-space and $\Gamma \vdash (\delta,\tau,\rho_{sz},t)$.*

(1) *If $pc =$ **static**, we have that $\mu = \mathbf{P}[\![E]\!]^0_{(\tau,\delta)}$ is a probability measure on $\mathbf{T}[\![T]\!]_{\rho_{sz}}$.*

(2) *If* $pc = \textsf{inst}$ *and* $0 \leq i \leq \rho_{sz}(t) - 1$, *we have that* $\mu = \mathbf{P}[\![E]\!]^{i}_{(\tau,\delta)}$ *is a probability measure on* $\mathbf{T}[\![T]\!]_{\rho_{sz}}$.
(3) *If* T *is in* $\textsf{det}$*-space there is closed* V *such that* $\mu = \texttt{return}\, V$.

Proof: By induction on the derivation of $\Gamma \vdash^{pc} E : T$, with appeal to lemma 10. ■

Semantics of Table $\mathbb{T}$ as a Measure $\boldsymbol{P}[\![\mathbb{T}]\!]^{t,\tau_{in}}_{(\tau,\delta)}$:

$\boldsymbol{P}[\![[]]\!]^{t,\tau_{in}}_{(\tau,\delta)} \triangleq \texttt{return}\,()$
$\boldsymbol{P}[\![(c : T\ \ell\ viz\ E) :: \mathbb{T}]\!]^{t,\tau_{in}}_{(\tau,\delta)} \triangleq \boldsymbol{P}[\![\mathbb{T}]\!]^{t,\tau_{in}}_{(\tau,\delta)}$ **if** $\text{space}(T) = \textsf{qry}$
$\boldsymbol{P}[\![(c : T\ \ell\ \textsf{input}\ \varepsilon) :: \mathbb{T}]\!]^{t,\tau_{in}}_{(\tau,\delta)} \triangleq \boldsymbol{P}[\![\mathbb{T}]\!]^{t,\tau}_{(\tau@[c\to\tau_{in}(c)],\delta)}$
$\boldsymbol{P}[\![(c : T\ \textsf{static local}\ E) :: \mathbb{T}]\!]^{t,\tau_{in}}_{(\tau,\delta)} \triangleq$
  $\boldsymbol{P}[\![E]\!]^{0}_{(\tau,\delta)} \gg\!= \lambda x.\boldsymbol{P}[\![\mathbb{T}]\!]^{t,\tau_{in}}_{(\tau@[c\to\textsf{static}(x)],\delta)} \gg\!= \lambda y.\texttt{return}\,(x,y)$
$\boldsymbol{P}[\![(c : T\ \textsf{static output}\ E) :: \mathbb{T}]\!]^{t,\tau_{in}}_{(\tau,\delta)} \triangleq$
  $\boldsymbol{P}[\![E]\!]^{0}_{(\tau,\delta)} \gg\!= \lambda x.([\tau_{in}(c) \preceq x]\boldsymbol{P}[\![\mathbb{T}]\!]^{t,\tau_{in}}_{(\tau@[c\to\textsf{static}(x)],\delta)}) \gg\!= \lambda y.\texttt{return}\,(x,y)$
$\boldsymbol{P}[\![(c : T\ \textsf{inst local}\ E) :: \mathbb{T}]\!]^{t,\tau_{in}}_{(\tau,\delta)} \triangleq$
  $\mu \gg\!= \lambda x.\boldsymbol{P}[\![\mathbb{T}]\!]^{t,\tau_{in}}_{(\tau@[c\to\textsf{inst}(x)],\delta)} \gg\!= \lambda y.\texttt{return}\,(x,y)$
  (where $\mu = \texttt{sequence}([\textsf{for}\ i < \rho_{sz}(t) \to \boldsymbol{P}[\![E]\!]^{i}_{(\tau,\delta)}])$)
$\boldsymbol{P}[\![(c : T\ \textsf{inst output}\ E) :: \mathbb{T}]\!]^{t,\tau_{in}}_{(\tau,\delta)} \triangleq$
  $\mu \gg\!= \lambda x.([\tau_{in}(c) \preceq x]\boldsymbol{P}[\![\mathbb{T}]\!]^{t,\tau_{in}}_{(\tau@[c\to\textsf{static}(x)],\delta)}) \gg\!= \lambda y.\texttt{return}\,(x,y)$
  (where $\mu = \texttt{sequence}([\textsf{for}\ i < \rho_{sz}(t) \to \boldsymbol{P}[\![E]\!]^{i}_{\tau}]$

Lemma 13. *If* $\Gamma \vdash^{\textsf{inst}} \mathbb{T} : Q$ *and* $\textsf{table}(Q)$ *and* $\Gamma \vdash (\delta,\tau,\rho_{sz},t)$ *and* $(t \mapsto \tau) \models^{\rho_{sz}}_{\textsf{in}} Q$, *then* $\mathbf{P}[\![\mathbb{T}]\!]^{t,\tau_{in}}_{(\tau,\delta)}$ *is a subprobability measure on* $\mathbf{T}[\![\mathbb{T}]\!]^{\rho_{sz}}_{t}$.

Proof: By induction on the derivation of $\Gamma \vdash^{\textsf{inst}} \mathbb{T} : Q$, with appeal to lemma 10. ■

Finally, to define the probabilistic semantics of Tabular schemas, we first need additional definitions: an operator `Bind`, which takes a (concrete) table τ and its description $\mathbb{T}$ and a list of values (represented as a nested pair Vs) and returns a map $\tau' = \texttt{Bind}(\mathbb{T}, Vs, \tau)$, obtained from τ by adding a corresponding value to every non-**qry** local or output attribute.

Auxiliary definition: $\texttt{Bind}(\mathbb{T}, Vs, \tau) = \tau'$

$\texttt{Bind}([], (), \tau) \triangleq []$
$\texttt{Bind}((c : T\ \ell\ \textsf{input}\ \varepsilon) :: \mathbb{T}, (V, Vs), \tau) \triangleq (c \mapsto \tau(c)) :: \texttt{Bind}(\mathbb{T}, Vs, \tau)$
$\texttt{Bind}((c : T\ \ell\ viz\ E) :: \mathbb{T}, (V, Vs), \tau) \triangleq \texttt{Bind}(\mathbb{T}, Vs, \tau)$ if $\text{space}(T) = \textsf{qry}$
$\texttt{Bind}((c : T\ \ell\ viz\ E) :: \mathbb{T}, (V, Vs), \tau) \triangleq (c \mapsto \ell\ V) :: \texttt{Bind}(\mathbb{T}, Vs, \tau)$ if $\text{space}(T) \neq \textsf{qry}$

Semantics of Schema $\mathbb{S}$ as a Measure $\boldsymbol{P}[\![\mathbb{S}]\!]^{\delta_{in}}_{\delta}$:

$\boldsymbol{P}[\![[]]\!]^{\delta_{in}}_{\delta} \triangleq \texttt{return}\,()$
$\boldsymbol{P}[\![(t = \mathbb{T}) :: \mathbb{S}]\!]^{\delta_{in}}_{\delta} \triangleq \boldsymbol{P}[\![\mathbb{T}]\!]^{t,\delta_{in}(t)}_{([],\delta)} \gg\!= \lambda x.\boldsymbol{P}[\![\mathbb{S}]\!]^{\delta_{in}}_{\delta@[t\to(\texttt{Bind}(\mathbb{T},x,\delta_{in}(t)))]} \gg\!= \lambda y.\texttt{return}\,(x,y)$

Lemma 14. *If $\Gamma \vdash \mathbb{S} : Sty$ and $\Gamma \vdash (\delta, \rho_{sz})$ and $(\delta_{in}, \rho_{sz}) \models^{\mathsf{in}} Sty$, then $\mathbf{P}[\![\mathbb{S}]\!]^{\delta_{in}}_{(\delta)}$ is a subprobability measure on $\mathbf{T}[\![\mathbb{S}]\!]^{\rho_{sz}}$.*

Proof: By induction on the derivation of $\Gamma \vdash \mathbb{S} : Sty$ with appeal to lemma 10. ■

E.3 Query semantics

In the previous subsection, we have shown how to compute a measure from a schema and a database. Here, we show how we can use this measure to evaluate queries included in this schema.

As before, we first need to define auxiliary operators and data structures. Here, marg returns the marginal for the given component of the measurable space norm simply normalizes a measure (we assume $|\mu| \neq 0$ and consider the measure not well-defined otherwise).

Auxiliary definitions

$\mathsf{marg}(\mu, i, n) \triangleq \mu \gg= \lambda(x_0, (\ldots, (x_{n-1}, ())\ldots).\mathtt{return}\, x_i$
$\mathsf{norm}(\mu) = \frac{\mu}{|\mu|}$

Additional data structures used in the query semantics

$\sigma ::= [t_i \mapsto \rho_i{}^{i \in 1..n}]$	map of marginals for a schema
$\rho ::= [c_i \mapsto q_i{}^{i \in 1..n}]$	map of marginals for a table
$q ::= \ell(\mu) \mid \ell([\mu_1, \ldots, \mu_n])$	tagged measure or array of measures

In order to evaluate queries in a schema with respect to a measure, we first need to compute the marginals of that measure for all the rows of measurable columns in that schema. Below, we present an algorithm for marginalizing a measure, and throughout this section we assume that the input σ to the query semantics is the map of the marginals.

The function measurables, used in the algorithm, returns an ordered list of names of columns whose domains are parts of the measurable space of the given table, that is, all **local** and **output** attributes not at level **qry**.

Measurable columns of a table:

$\mathsf{measurables}([]) \triangleq []$
$\mathsf{measurables}((c : T\ \ell\ \mathbf{input}\ \varepsilon) :: \mathbb{T}) \triangleq \mathsf{measurables}(\mathbb{T})$
$\mathsf{measurables}((c : T\ \ell\ viz\ E) :: \mathbb{T}) \triangleq \mathsf{measurables}(\mathbb{T})$ if $\mathsf{space}(T) = \mathbf{qry}$
$\mathsf{measurables}((c : T\ \ell\ viz\ E) :: \mathbb{T}) \triangleq c :: \mathsf{measurables}(\mathbb{T})$ if $\mathsf{space}(T) \neq \mathbf{qry}$

The algorithm also makes use of a simple auxiliary function isInst returning true if and only if a given column is at **inst** level in the given table. Its formal definition is ommitted, but would be straightforward to write down. To keep the notation cleaner, we do not make explicit the dependence of m on i.

Measure marginalization: $\mathsf{marginalize}(\mathbb{S},\rho_{sz},\mu)$

```
(1) Assume Core($\mathbb{S}$) and $\mathbb{S} = [(t_1 = \mathbb{T}_1),\ldots,(t_n = \mathbb{T}_n)]$
(2) For all $i \in 1..n$:
    Let $\mu_i = \mathsf{marg}(\mu, i-1, n)$ and $[c_1,\ldots,c_m] = \mathsf{measurables}(\mathbb{T}_i)$
    (3) For all $j \in 1..m$:
        Let $\mu_{ij} = \mathsf{norm}(\mathsf{marg}(\mu_i, j-1, m))$
        If $\mathsf{isInst}(\mathbb{T}_i, c_j)$ then:
            $q_{ij} = \mathbf{inst}([v_0,\ldots,v_{\rho_{sz}(t)-1}])$, where $v_k = \mathsf{marg}(\mu_{ij}, k, \rho_{sz}(t))$
        Else:
            $q_{ij} = \mathbf{static}(\mu_{ij})$
    (4) Let $\rho_i = [c_j \mapsto q_{ij}\ ^{j\in 1..m}]$
(5) Let $\sigma = [t_i \mapsto \rho_i\ ^{i\in i..n}]$
(6) Return $\sigma$.
```

With these definitions in place, we can now define the query semantics for Tabular programs. To this end, we require rules for evaluating deterministic expressions as well as extracting the parameters of inferred distributions. We also define rules for evaluating tables, which create a map storing all deterministic expressions in a given table, as well as rules for evaluating schemas. The output of a schema evaluation contains valuations for all deterministic attributes of a schema, rather than just valuations for queries. This is purely for mathematical convenience—it would be straightforward to filter the output at the end.

Our query implementation depends on deterministic but approximate inference algorithms in Infer.NET. In our semantics of queries, we write $\mathsf{approx}(\mu, D, [V_1',\ldots,V_m'])$ as an abstraction of these algorithms, subject to the following assumption.

Assumption: Abstraction of Approximate Inference:

Suppose $D : [x_1 : T_1,\ldots,x_m : T_m](y_1 : U_1,\ldots,y_n : U_n) \to T$.
Suppose μ is a probability measure on $\boldsymbol{T}[\![T\{V_1'/x_1\}\ldots\{V_m'/x_m\}]\!]$.
We assume $\mathsf{approx}(\mu, D, [V_1',\ldots,V_m'])$ computes $[y_i \mapsto V_i\ ^{i\in 1..n}]$
such that the measure $\mu_{D[V_1',\ldots,V_m'](V_1,\ldots,V_n)}$ approximates μ.

In the definitions below, we assume an input database (δ_{in}, ρ_{sz}), as well as an environment $\hat{\sigma}$ mapping tables to corresponding marginal measures, constructed from the output of the probabilistic semantics.

Expression evaluation: $\delta;\rho;i \vdash E \Downarrow V$

(Query Infer Static Dist) (where $j \in 1..n$)

$$\frac{\delta;\tau;\rho;i \vdash e_k \Downarrow V_k' \quad \forall k \in 1..m \quad \rho(x) = \mathbf{static}(v) \qquad \mathsf{approx}(v, D, [V_1';\ldots;V_m']) = [y_k \mapsto V_k\ ^{k\in 1..n}]}{\delta;\tau;\rho;i \vdash \mathbf{infer}.D[e_1,\ldots,e_m].y_j(x) \Downarrow V_j}$$

(QUERY INFER INST DIST) (where $j \in 1..n$)

$$\frac{\delta;\tau;\rho;i \vdash e_k \Downarrow V'_k \quad \forall k \in 1..m \quad \rho(x) = \textbf{inst}([v_0,\dots,v_{l-1}]) \qquad \mathsf{approx}(v_i, D, [V'_1;\dots;V'_m]) = [y_k \mapsto V_k\ {}^{k\in 1..n}]}{\delta;\tau;\rho;i \vdash \textbf{infer}.D[e_1,\dots,e_m].y_j(x) \Downarrow V_j}$$

(QUERY VAR STATIC)

$$\frac{\tau(x) = \textbf{static}(V)}{\delta;\tau;\rho;i \vdash x \Downarrow V}$$

(QUERY VAR INST)

$$\frac{\tau(x) = \textbf{inst}([V_0,\dots V_{n-1}])}{\delta;\tau;\rho;i \vdash x \Downarrow V_i}$$

(QUERY CONST)

$$\frac{}{\delta;\tau;\rho;i \vdash s \Downarrow s}$$

(QUERY PRIM)

$$\frac{\delta;\tau;\rho;i \vdash E_j \Downarrow V_j \quad \forall j \in 1..n}{\delta;\tau;\rho;i \vdash g(E_1,\dots,E_n) \Downarrow \underline{g}(V_1,\dots,V_n)}$$

(QUERY IF TRUE)

$$\frac{\delta;\tau;\rho;i \vdash E_1 \Downarrow \textbf{true} \quad \delta;\tau;\rho;i \vdash E_2 \Downarrow V}{\delta;\tau;\rho;i \vdash \textbf{if } E_1 \textbf{ then } E_2 \textbf{ else } E_3 \Downarrow V}$$

(QUERY IF FALSE)

$$\frac{\delta;\tau;\rho;i \vdash E_1 \Downarrow \textbf{false} \quad \delta;\tau;\rho;i \vdash E_3 \Downarrow V}{\delta;\tau;\rho;i \vdash \textbf{if } E_1 \textbf{ then } E_2 \textbf{ else } E_3 \Downarrow V}$$

(QUERY ARRAY)

$$\frac{\delta;\tau;\rho;i \vdash E_j \Downarrow V_j \quad \forall j \in 0..n-1}{\delta;\tau;\rho;i \vdash [E_0,\dots,E_{n-1}] \Downarrow [V_0,\dots,V_{n-1}]}$$

(QUERY INDEX) (where $j \in 0..n-1$)

$$\frac{\delta;\tau;\rho;i \vdash E \Downarrow [V_0,\dots,V_{n-1}] \quad F \Downarrow j}{\delta;\tau;\rho;i \vdash E[F] \Downarrow V_j}$$

(QUERY ITER)

$$\frac{\delta;\tau;\rho;i \vdash F\{{}^{j}\!/_{x}\} \Downarrow V_j \quad \forall j \in 0..n-1}{\delta;\tau;\rho;i \vdash [\textbf{for } x < n \rightarrow F] \Downarrow [V_0,\dots,V_{n-1}]}$$

(QUERY DEREF STATIC)

$$\frac{\delta(t)(c) = \textbf{static } V}{\delta;\tau;\rho;i \vdash t.c \Downarrow V}$$

(QUERY DEREF INST)

$$\frac{\delta;\tau;\rho;i \vdash E \Downarrow k \qquad \delta(t)(c) = \textbf{inst}[V_0,\dots,V_{\rho_{sz}(t)-1}]}{\delta;\tau;\rho;i \vdash E : t.c \Downarrow V_k}$$

(QUERY SIZEOF)

$$\frac{\rho_{sz}(t) = n}{\delta;\tau;\rho;i \vdash \textbf{sizeof}(t) \Downarrow n}$$

Table Evaluation: $t;\delta;\tau_{in};\rho \vdash \mathbb{T} \Downarrow \tau$

(VAL EMPTY)

$$\frac{}{t;\delta;\tau_{in};\rho \vdash [] \Downarrow []}$$

(VAL INPUT)

$$\frac{t;\delta;\tau_{in},(c \rightarrow \delta_{in}(t)(c));\rho \vdash \mathbb{T} \Downarrow \tau}{t;\delta;\tau_{in};\rho \vdash (c : T\ \ell\ \textbf{input}\ \varepsilon) :: \mathbb{T} \Downarrow \tau@[c \mapsto \delta_{in}(t)(c)]}$$

(VAL RANDOM) (where space$(T) = \textbf{rnd}$)

$$\frac{t;\delta;\tau_{in};\rho \vdash \mathbb{T} \Downarrow \tau}{t;\delta;\tau_{in};\rho \vdash (c : T\ \ell\ viz\ E) :: \mathbb{T} \Downarrow \tau}$$

(VAL QUERYORDET STATIC) (where space$(T) \neq \textbf{rnd}$)

$$\frac{\delta;\tau_{in};\rho;0 \vdash E \Downarrow V \qquad t;\delta;\tau_{in},(c \rightarrow \textbf{static}(V));\rho \vdash \mathbb{T} \Downarrow \tau}{t;\delta;\tau_{in};\rho \vdash (c : T\ \textbf{static}\ viz\ E) :: \mathbb{T} \Downarrow \tau@[c \mapsto \textbf{static}\ V]}$$

(VAL QUERYORDET INST) (where space$(T) \neq \textbf{rnd}$)

$$\frac{\delta;\tau_{in};\rho(t);i \vdash E \Downarrow V_i \quad \forall i \in 0..(\rho_{sz}(t)-1) \qquad t;\delta;\tau_{in},(c \rightarrow \textbf{inst}([V_0,\dots,V_{\rho_{sz}(t)-1}]));\rho \vdash \mathbb{T} \Downarrow \tau}{t;\delta;\tau_{in};\rho \vdash (c : T\ \textbf{inst}\ viz\ E) :: \mathbb{T} \Downarrow \tau@[c \mapsto \textbf{inst}([V_0,\dots,V_{\rho_{sz}(t)-1}])]}$$

Schema evaluation: $\delta \vdash_\sigma \mathbb{S} \Downarrow \delta_{out}$

(QUERY SCHEMA EMPTY)

$$\frac{}{\delta \vdash_\sigma [] \Downarrow []}$$

(QUERY SCHEMA TABLE)

$$\frac{t; \delta; \varnothing; \sigma(t) \vdash \mathbb{T} \Downarrow \tau_t \qquad \delta, (t \to \tau_t) \vdash_\sigma \mathbb{S} \Downarrow \delta_{out}}{\delta \vdash_\sigma (t = \mathbb{T}) :: \mathbb{S} \Downarrow \delta_{out} @ [t \to \tau_t]}$$

E.4 Algorithm for Query Evaluation and Conformance Relation

By combining the query evaluation rules with the probabilistic semantics, we can fully define an algorithm for computing a map of query results from a schema, with respect to a given database:

Algorithm 2: Query Semantics of Core Schema: $\mathsf{CoreQuery}(\mathbb{S}, DB)$

(1) Assume $\mathsf{core}(\mathbb{S})$ and $DB = (\delta_{in}, \rho_{sz})$.
(2) Let $\mu \triangleq \boldsymbol{P}[\![\mathbb{S}]\!]_{[]}^{\delta_{in}}$ (that is, the joint distribution over all **rnd**-variables).
(3) Let $\sigma = \mathsf{marginalize}(\mathbb{S}, \rho_{sz}, \mu)$.
(4) Return $(\delta_{out}, \rho_{sz})$ such that $\varnothing \vdash_\sigma \mathbb{S} \Downarrow \delta_{out}$.

In order to prove the key theoretical result about the evaluation algorithm, we need three more conformance relations. The judgment $\rho \vdash_t^{\rho_{sz}} \mathbb{T}$ says that the map of marginals ρ contains measures corresponding to all columns of $\mathbb{T}$ named t, and that these measures are subprobability measures on the right measurable spaces. The statement $\sigma \vdash^{\rho_{sz}} \mathbb{S}$ states that all tables in $\mathbb{S}$ have corresponding maps of marginals in σ. Finally $\Gamma \vdash_t^{\rho_{sz}} \rho$ is similar to the first judgment, except that it checks conformance of ρ to a typing environment.

New conformance rules

$\rho \vdash_t^{\rho_{sz}} \mathbb{T}$	ρ conforms to $\mathbb{T}$
$\sigma \vdash^{\rho_{sz}} \mathbb{S}$	σ conforms to $\mathbb{S}$
$\Gamma \vdash_t^{\rho_{sz}} \rho$	ρ makes sense in Γ

Conformance Rules for Maps of Marginals : $\rho \vdash_t^{\rho_{sz}} \mathbb{T}$**,** $\sigma \vdash^{\rho_{sz}} \mathbb{S}$

(MARG SCHEMA)

$$\frac{\sigma(t_i) \vdash_{t_i}^{\rho_{sz}} \mathbb{T}_i \quad \forall i \in 1..n}{\sigma \vdash^{\rho_{sz}} [(t_1 = \mathbb{T}_1), \dots, (t_n = \mathbb{T}_n)]}$$

(MARG TABLE)

$$\frac{\begin{array}{l} I_S = \{ j \in 1..m \mid \mathrm{space}(T_j) = \mathbf{rnd} \wedge \ell_j = \mathbf{static} \} \\ I_I = \{ j \in 1..m \mid \mathrm{space}(T_j) = \mathbf{rnd} \wedge \ell_j = \mathbf{inst} \} \\ \rho(c_j) = \mathbf{static}(\mu_j) \text{ and } \mu_j \text{ is a subprobability measure on } \boldsymbol{T}[\![T_j]\!]_{\rho_{sz}}^{\rho_{sz}(t)} \quad \forall j \in I_S \\ \rho(c_j) = \mathbf{inst}([\mu_{j,0}, \dots, \mu_{j,\rho_{sz}(t)-1}]) \text{ and} \\ \qquad \mu_{j,0}, \dots, \mu_{j,\rho_{sz}(t)-1} \text{ are subprobability measures on } \boldsymbol{T}[\![T_j]\!]_{\rho_{sz}}^{\rho_{sz}(t)} \quad \forall j \in I_I \end{array}}{\rho \vdash_t^{\rho_{sz}} [(c_j : T_j \ \ell_j \ viz_j)^{\ j \in 1..m}]}$$

(CONF MARG)

$$\frac{\begin{array}{l} I_S = \{j \in 1..n \mid \text{space}(T_j) = \textbf{rnd} \wedge \ell_j = \textbf{static}\} \\ I_I = \{j \in 1..n \mid \text{space}(T_j) = \textbf{rnd} \wedge \ell_j = \textbf{inst}\} \\ \rho(c_j) = \textbf{static}(\mu_j) \text{ and } \mu_j \text{ is a subprobability measure on } \boldsymbol{T}[\![T_j]\!]^{\rho_{sz}(t)}_{\rho_{sz}} \quad \forall j \in I_S \\ \rho(c_j) = \textbf{inst}([\mu_{j,0}, \dots, \mu_{j,\rho_{sz}(t)-1}) \text{ and} \\ \quad \mu_{j,0}, \dots, \mu_{j,\rho_{sz}(t)-1} \text{ are subprobability measures on } \boldsymbol{T}[\![T_j]\!]^{\rho_{sz}(t)}_{\rho_{sz}} \quad \forall j \in I_I \end{array}}{(t_i : Q_i)^{\ i \in 1..m}, (c_j :^{\ell_j} T_j)^{\ j \in 1..n} \vdash^{\rho_{sz}}_{t} \rho}$$

To prove that CoreQuery returns a database conforming to the type of the given schema for a well-formed input, we first need to show that the reduction of the queries in the schema actually terminates and returns a result. To this end, we need the following three lemmas, which are analogues for progress lemmas for reduction systems.

Lemma 15. – *If $\Gamma \vdash^{\textsf{static}} E : T$ and $\Gamma \vdash (\delta, \tau, \rho_{sz}, t)$ and $\Gamma \vdash^{\rho_{sz}}_{t} \rho$, then there is a V such that $\delta; \rho; 0 \vdash E \Downarrow V$.*

– *If $\Gamma \vdash^{\textsf{inst}} E : T$ and $\Gamma \vdash (\delta, \tau, \rho_{sz}, t)$ and $\Gamma \vdash^{\rho_{sz}}_{t} \rho$ and $i \in 0..\rho_{sz}(t) - 1$, then there is a V such that $\delta; \rho; i \vdash E \Downarrow V$.*

Proof: By induction on the derivation of $\Gamma \vdash^{\textsf{static}} E : T$ and $\Gamma \vdash^{\textsf{inst}} E : T$. ■

Lemma 16. *If $\Gamma \vdash^{\textsf{inst}} \mathbb{T} : Q$ and $\Gamma \vdash (\delta, \tau, \rho_{sz}, t)$ and $\Gamma \vdash^{\rho_{sz}}_{t} \rho$ and $\rho \vdash^{\rho_{sz}}_{t} \mathbb{T}$, then there is a τ such that $t; \delta; \tau; \rho \vdash \mathbb{T} \Downarrow \tau$.*

Proof: By induction on the derivation of $\Gamma \vdash^{\textsf{inst}} \mathbb{T} : Q$, with appeal to Lemma 15. ■

Lemma 17. *If $\Gamma \vdash \mathbb{S} : Sty$ and $\Gamma \vdash (\delta, \rho_{sz})$ and $\sigma \vdash^{\rho_{sz}} \mathbb{S}$, then there exists a δ_{out} such that $\delta \vdash_{\sigma} \mathbb{S} \Downarrow \delta_{out}$.*

Proof: By induction on the derivation of $\Gamma \vdash \mathbb{S} : Sty$, with appeal to Lemma 16. ■

Subsequently, we have to prove that if a schema can be evaluated into an output map, then this map matches the type of the schema. To show that, we need three more lemmas, akin to the standard type preservation results.

Lemma 18. *If $\Gamma \vdash^{\ell} E : T$ and $\delta, \tau; \rho; i \vdash E \Downarrow V$,then $\varnothing \vdash^{\ell} V : [\![T]\!]^{\ell,t}_{\rho_{sz}}$.*

Proof: By induction on the derivation of $\Gamma \vdash^{\ell} E : T$. ■

Lemma 19. *If $\Gamma \vdash^{\textsf{inst}} \mathbb{T} : Q$ and $t; \delta; \tau; \sigma \vdash \mathbb{T} \Downarrow \tau$,then $t \mapsto \tau \models^{\textsf{out}}_{\rho_{sz}} Q$.*

Proof: By induction on the derivation of $\Gamma \vdash^{\textsf{inst}} \mathbb{T} : Q$, with appeal to lemma 18. ■

Lemma 20. *If $\Gamma \vdash \mathbb{S} : Sty$ and $\delta \vdash_{\sigma} \mathbb{S} \Downarrow \delta_{out}$ and $\Gamma \vdash (\delta, \rho_{sz})$, then $(\delta_{out}, \rho_{sz}) \models^{\textsf{out}} Sty$.*

Proof: By induction on the derivation of $\Gamma \vdash \mathbb{S} : Sty$, with appeal to lemma 19. ■

Finally, we have to show that marginalize returns a map of marginals conforming to the database schema.

Lemma 21. *If μ is a subprobability measure on $\mathbf{T}[\![\mathbb{S}]\!]^{\rho_{sz}}$, then there exists a unique σ such that $\sigma = \mathsf{marginalize}(\mathbb{S}, \rho_{sz}, \mu)$. Moreover, $\sigma \vdash^{\rho_{sz}} \mathbb{S}$.*

Proof: The existence and uniqueness of σ follow from the fact that marginalize is deterministic and marginalization is well-defined.

The result $\sigma \vdash^{\rho_{sz}} \mathbb{S}$ follows from the definition of the measurable space. ■

Restatement of Theorem 2 *Suppose* $\mathsf{Core}(\mathbb{S})$ *and* $\varnothing \vdash \mathbb{S} : Sty$ *and* $DB = (\delta_{in}, \rho_{sz})$ *and* $DB \models^{\mathsf{in}} Sty$*. Then algorithm* $\mathsf{CoreQuery}(\mathbb{S}, DB)$ *returns a unique* $DB' = (\delta_{out}, \rho_{sz})$ *such that* $DB' \models^{\mathsf{out}} Sty$*.*

Proof: Since we have $\varnothing \vdash \mathbb{S} : Sty$ and $\delta_{in} \models^{\mathsf{in}}_{\rho_{sz}} Sty$ by assumption, and $\varnothing \vdash ([], \rho_{sz})$ follows trivially, Lemma 14 says that $\boldsymbol{P}[\![\mathbb{S}]\!]^{\delta_{in}}_{[]}$ is a subprobability measure on $\boldsymbol{T}[\![\mathbb{S}]\!]^{\rho_{sz}}$. Thus, by Lemma 21, there is a unique σ such that $\sigma = \mathsf{marginalize}(\mathbb{S}, \rho_{sz}, \mu)$, which satisfies $\sigma \vdash^{\rho_{sz}} \mathbb{S}$. By Lemma 17 there exists a δ_{out} such that $\varnothing \vdash_{\sigma} \mathbb{S} \Downarrow \delta_{out}$. This δ_{out} is unique, since the algorithm CoreQuery is deterministic. This shows that the algorithm terminates and returns a unique output database.

The desired conformance result $DB' \models^{\mathsf{out}} Sty$ for $DB' = (\delta_{out}, \rho_{sz})$ follows directly from Lemma 20. ■

F Proofs for Section 5

We give proofs for Proposition 1, Proposition 2, and Proposition 3.

F.1 Basic properties of the type system

Lemma 22 (Scoping).

(1) *If $\Gamma \vdash^{pc} M : Q$ then* $\mathrm{fv}(M) \subseteq \mathrm{dom}(\Gamma)$ *and* $\mathrm{fv}(Q) \subseteq \mathrm{dom}(\Gamma)$.
(2) *If $\Gamma \vdash \mathbb{T} : Q$ then* $\mathrm{fv}(\mathbb{T}) \subseteq \mathrm{dom}(\Gamma)$ *and* $\mathrm{fv}(Q) \subseteq \mathrm{dom}(\Gamma)$.
(3) *If $\Gamma \vdash \mathbb{S} : Sty$ then* $\mathrm{fv}(\mathbb{S}) \subseteq \mathrm{dom}(\Gamma)$ *and* $\mathrm{fv}(Sty) \subseteq \mathrm{dom}(\Gamma)$.

Proof: The proof is by induction on the depth of the derivations. ■

Lemma 23 (Derived Judgments).

(1) *If $\Gamma \vdash T$ then $\Gamma \vdash \diamond$.*
(2) *If $\Gamma \vdash^{pc} E : T$ then $\Gamma \vdash T$.*
(3) *If $\Gamma \vdash^{pc} M : Q$ then $\Gamma \vdash Q$.*

(4) *If* $\Gamma \vdash \mathbb{T} : Q$ *then* $\Gamma \vdash Q$.
(5) *If* $\Gamma \vdash^{pc} R : Q \to Q'$ *then* $\Gamma \vdash Q$ *and* $\Gamma \vdash Q'$.
(6) *If* $\Gamma \vdash \mathbb{S} : Sty$ *then* $\Gamma \vdash Sty$.

Proof: By induction on the depth of the derivations. ■

Lemma 24. *If* $\Gamma \vdash^{pc} \mathbb{T} : Q$ *then* $\mathrm{dom}(\Gamma) \cap \mathrm{dom}(Q) = \varnothing$.

Proof: By induction on the derivation of $\Gamma \vdash^{pc} \mathbb{T} : Q$ ■

Lemma 25. *If* $\Gamma \vdash^{pc} E : T$ *and* $x \in \mathrm{fv}(E)$ *then* $(x :^{\ell} T') \in \Gamma$ *with* $\ell \leq pc$ *for some* ℓ *and* T'.

Proof: By induction on the derivation of $\Gamma \vdash^{pc} E : T$. ■

Subtyping is reflexive and transitive:

Lemma 26.

(1) *If* $\Gamma \vdash T$ *then* $\Gamma \vdash T <: T$.
(2) *If* $\Gamma \vdash T <: T'$ *and* $\Gamma \vdash T' <: T''$ *then* $\Gamma \vdash T <: T''$.

Proof: By induction on the derivations. ■

Since every column has a type annotation, tables and schemas have unique types, if they exist.

Lemma 27 (Unique Types).

(1) *If* $\Gamma \vdash \mathbb{T} : Q$ *and* $\Gamma \vdash \mathbb{T} : Q'$ *then* $Q = Q'$.
(2) *If* $\Gamma \vdash \mathbb{S} : Sty$ *and* $\Gamma \vdash \mathbb{S} : Sty'$ *then* $Sty = Sty'$.

Proof: By inductions on the derivations. ■

F.2 Proof of Proposition 1

First, we need some additional definitions and lemmas:

Converting Table Types to Environments: $\gamma(Q)$, $\gamma(\mathbb{T})$

$$\gamma([\,]) \triangleq \varnothing \qquad \gamma((c : T\ \ell\ viz) :: Q) \triangleq c :^{\ell} T, \gamma(Q) \qquad \gamma((c : T\ (\ell\ viz)\ E) :: \mathbb{T}) \triangleq c :^{\ell} T, \gamma(\mathbb{T})$$

Definition 1. *We say that* $\Gamma_2 \leq \Gamma_1$ *if and only if* $\mathrm{dom}(\Gamma_2) \subseteq \mathrm{dom}(\Gamma_1)$ *and all variables in* $\mathrm{dom}(\Gamma_2)$ *have levels not lower in* Γ_1 *than in* Γ_2.

Lemma 28. *If* $\mathrm{Core}(\mathbb{T})$ *and* $\Gamma_2 \vdash \diamond$ *and* $\Gamma_1 \vdash^{pc} \mathbb{T} : Q$ *and* $\mathrm{dom}(\Gamma_2) \cap \mathrm{dom}(\mathbb{T}) = \varnothing$ *and* $\Gamma_1 \leq \Gamma_2$*, then* $\Gamma_2 \vdash^{pc} \mathbb{T} : Q$.

Proof: By induction on the depth of derivation of $\Gamma_1 \vdash^{pc} \mathbb{T} : Q$. ■

Lemma 29. *If* Core($\mathbb{T}$) *and* $\Gamma \vdash^{pc} \mathbb{T} : Q$*, then* $\gamma(Q) \leq \gamma(\mathbb{T} \wedge pc)$.

Proof: By induction on the depth of derivation of $\Gamma \vdash^{pc} \mathbb{T} : Q$. ■

Lemma 30. (1) *If* $\Gamma \vdash^{pc} E : T$ *and* $x \notin \Gamma$*, and* $\Gamma \vdash U$*, then* $\Gamma, x :^{\ell} U \vdash^{pc} E : T$.
(2) *If* $\Gamma \vdash^{pc} R : Q \to Q'$ *and* $x \notin \Gamma$*, and* $\Gamma \vdash U$*, then* $\Gamma, x :^{\ell} U \vdash^{pc} R : Q \to Q'$

Proof:

(1) By induction on the derivation of $\Gamma \vdash^{pc} E : T$.
(2) By induction on the derivation of $\Gamma \vdash^{pc} R : Q \to Q'$.

■

Lemma 31. *If* $\Gamma, c :^{\ell \wedge pc} T \vdash^{pc} \mathbb{T} : Q$ *and* $\Gamma \vdash^{\ell \wedge pc} e : T$*, then* $\Gamma \vdash^{pc} \mathbb{T}\{^{e}/_{c}\} : Q\{^{e}/_{c}\}$

Proof: Follows directly from the previous lemma ■

Lemma 32. *Appending Columns to Tables*
If $\Gamma \vdash^{pc} (c : T\ (\ell\ viz)\ E) : Q_1$ *and* $\Gamma, c :^{\ell \wedge pc} T \vdash^{pc} \mathbb{T} : Q_2$*, then* $\Gamma \vdash^{pc} (c : T\ (\ell\ viz)\ E) :: \mathbb{T} : Q_1 @ Q_2$.

Proof: By case analysis on the typechecking rules for tables. Q_1 obtained by using an empty table as tail. ■

Lemma 33. *Type of Multiple Concatenated Columns*
If $\Gamma, c_1 :^{\ell_1 \wedge pc} T_1, \ldots, c_{i-1} :^{\ell_{i-1} \wedge pc} T_{i-1} \vdash^{pc} (c_i : T_i\ (\ell_i\ viz_i)\ M_i) : Q_i$ *for all* $i \in 1 \ldots n$ *and* $\Gamma, c_1 :^{\ell_1 \wedge pc} T_1, \ldots, c_{i-1} :^{\ell_i \wedge pc} T_i \vdash^{pc} \mathbb{T} : Q$*, then* $\Gamma \vdash^{pc} [(c_i : T_i\ (\ell_i\ viz_i)\ M_i)^{i \in 1 \ldots n}] @ \mathbb{T} : Q_i^{i \in 1 \ldots n} @ Q$

Proof: By induction on n, with appeal to Lemma 32. ■

Lemma 34. *If* $(\mathbb{T} \wedge \textsf{local}) \vdash R \rightsquigarrow_o \mathbb{T}_1$ *and* $\Gamma \vdash^{\ell \wedge pc} \mathbb{T}\ R : Q$*, then* $\Gamma \vdash^{pc} \mathbb{T}_1 : [\,]$

Proof: By definition of $\mathbb{T} \wedge viz$, all columns of $\mathbb{T} \wedge$ **local** must have visibility **input** or **local**. As applying a function to arguments converts all its **input** columns to **local** ones, and leaves **local** columns unchanged, all columns of $\mathbb{T}_1$ must be at level **local**.

The statement $\Gamma \vdash^{\ell \wedge pc} \mathbb{T}\ R : Q$ must have been derived using (MODEL APPL):

(MODEL APPL)

$$\frac{\Gamma \vdash^{\ell \wedge pc} \mathbb{T} : Q_1 \qquad \textsf{fun}(Q_1) \qquad \Gamma \vdash^{\ell \wedge pc} R : Q_1 \to Q}{\Gamma \vdash^{\ell \wedge pc} (\mathbb{T}\ R) : Q}$$

Thus, the result $\Gamma \vdash^{pc} \mathbb{T}_1 : [\,]$ follows by simple induction: from $\Gamma \vdash^{\ell \wedge pc} R : Q_1 \to Q$, we get $\Gamma \vdash \diamond$, so by (TABLE EMPTY), $\Gamma \vdash^{pc} [\,] : [\,]$. Now, let $\mathbb{T}_1 = (c : T\ \ell\ \mathsf{local}\ E) :: \mathbb{T}''$. We get $\Gamma \vdash^{\ell} E : T$ from the derivation of either $\Gamma \vdash^{\ell \wedge pc} \mathbb{T} : Q_1$ or $\Gamma \vdash^{\ell \wedge pc} R : Q_1 \to Q$, depending on whether c is an input or not in $\mathbb{T}$. From $\Gamma \vdash^{\ell \wedge pc} \mathbb{T} : Q_1$, we also get $c \notin \Gamma$. By applying the induction hypothesis and lemma 30, we obtain $\Gamma \vdash^{\ell} E : T$ and $c \notin \Gamma$. Thus, by (TABLE LOCAL), we get $\Gamma \vdash^{pc} (c : T\ \ell\ \mathsf{local}\ E) :: \mathbb{T} : [\,]$.

■

Lemma 35. *If $\Gamma \vdash^{pc} \mathbb{T} : Q$, then $\Gamma \vdash^{pc} \mathbb{T}[e_{index} < e_{size}] : Q[e_{size}]$.*

Proof: z By induction on the derivation of $\Gamma \vdash^{pc} \mathbb{T} : Q$, using the definitions of $\mathbb{T}[e_{index} < e_{size}]$ and $Q[e_{size}]$. ■

Notation for substituting column names: $(\mathbb{T})[a/b]$**,** $(Q)[a/b]$

$((\mathsf{b} : T\ (\ell\ viz)\ E) :: \mathbb{T})[a/b] \triangleq (\mathsf{a} : T\ (\ell\ viz)\ E) :: (\mathbb{T}[a/b])$
$((\mathsf{c} : T\ (\ell\ viz)\ E) :: \mathbb{T})[a/b] \triangleq (\mathsf{c} : T\ (\ell\ viz)\ E) :: (\mathbb{T}[a/b])$ if $c \neq b$
$((\mathsf{b} : T\ (\ell\ viz)) :: Q)[a/b] \triangleq (\mathsf{a} : T\ (\ell\ viz)) :: (Q[a/b])$
$((\mathsf{c} : T\ (\ell\ viz)) :: Q)[a/b] \triangleq (\mathsf{c} : T\ (\ell\ viz))) :: (Q[a/b])$ if $c \neq b$
$[][a/b] \triangleq []$

Lemma 36. *If $((\mathbb{T}_1 \wedge \ell) \wedge viz) \vdash R \rightsquigarrow_o \mathbb{T}_2$ ands $\Gamma \vdash^{\ell} (\mathbb{T}_1\ R) : Q$, then $\gamma(Q[o/\mathsf{ret}]) \leq \gamma(\mathbb{T}_2)$*

Proof: The judgment $\Gamma \vdash^{\ell} \mathbb{T}\ R : Q$ must have been derived using (MODEL APPL):

(MODEL APPL)
$$\frac{\Gamma \vdash^{\ell} \mathbb{T}_1 : Q' \quad \mathsf{fun}(Q') \quad \Gamma \vdash^{\ell} R : Q' \to Q}{\Gamma \vdash^{\ell} (\mathbb{T}_1\ R) : Q}$$

By lemma 29, $\gamma(Q') \leq \gamma(\mathbb{T}_1)$. By the definition of $\rightsquigarrow'$, we have $\gamma(((\mathbb{T}_1 \wedge \ell) \wedge viz)[o/\mathsf{ret}]) = \gamma(\mathbb{T}_2)$.

Also, note that $\mathrm{dom}(((\mathbb{T}_1 \wedge \ell) \wedge viz)[o/\mathsf{ret}]) = \mathrm{dom}(\mathbb{T}_1[o/\mathsf{ret}])$. Now, we just need to show that $\mathrm{dom}(Q[o/\mathsf{ret}]) \subseteq \mathrm{dom}(\mathbb{T}_1[o/\mathsf{ret}])$ and that levels of all columns in Q are greater or equal to levels of corresponding columns in $(\mathbb{T}_1 \wedge \ell) \wedge viz$. The first part is trivial, since Q' contains a subset of columns of $\mathbb{T}_1$, and by the typing rules for arguments, Q contains only the output columns of Q'. As for the levels, by table and argument checking rules, columns of Q have the levels of $\mathbb{T}_1$ with those greater then ℓ reduced to ℓ. Meanwhile, by definition of join, columns of $(\mathbb{T}_1 \wedge \ell) \wedge viz$ also have the same levels as those in the original table $\mathbb{T}_1$ with those greater than ℓ reduced to ℓ. This concludes the proof.

■

Lemma 37. *If* $\mathsf{Core}(\mathbb{T})$*,* $(\mathbb{T} \wedge \ell) \vdash R \rightsquigarrow_o \mathbb{T}_1$ *and* $\Gamma \vdash^{\ell} (\mathbb{T}\ R) : Q'$ *and the last column of* $\mathbb{T}$ *is at level* ℓ*, then* $\Gamma \vdash^{\ell} \mathbb{T}_1 : Q'[o/\mathsf{ret}]$.

Proof: By (MODEL APPL), this theorem can be rewritten as:

If $\mathsf{Core}(\mathbb{T})$, $(\mathbb{T} \wedge \ell) \vdash R \rightsquigarrow_o \mathbb{T}_1$ and $\Gamma \vdash^{\ell} \mathbb{T} : Q$ and $\Gamma \vdash^{\ell} R : Q \to Q'$ and $\mathrm{dom}(\mathbb{T}) \cap \mathrm{dom}(\Gamma) = \varnothing$ and $\mathbf{fun}(\mathbb{T})$ and the last column of $\mathbb{T}$ is at level ℓ, then $\Gamma \vdash \mathbb{T}_1 : Q'[o/\mathsf{ret}]$.

We prove the rewritten version by induction on the depth of derivation of $(\mathbb{T} \wedge \ell) \vdash R \rightsquigarrow_o \mathbb{T}_1$.

- Case:
  (APPLY OUTPUT)

  $$\frac{}{[(\mathsf{ret} : T\ (\ell\ viz)\ E)] \vdash [] \rightsquigarrow_o [(o : T\ (\ell\ viz)\ E)]}$$

  - Subcase $viz = \mathbf{output}$:
    (TABLE OUTPUT)(for Core tables)
    $$\frac{\Gamma_1 \vdash^{\ell} E : T}{\Gamma_1 \vdash^{\ell} (\mathsf{ret} : T\ \ell\ \mathbf{output}\ E) : (\mathsf{ret} : T\ \ell\ \mathbf{output})}$$
    (ARG OUTPUT)
    $$\frac{\Gamma \vdash T \qquad \Gamma \vdash^{\ell} R : [] \to []}{\Gamma \vdash^{\ell} R : (\mathsf{ret} : T\ (\ell\ \mathbf{output})) \to (\mathsf{ret} : T\ (\ell\ \mathbf{output}))}$$
    Here, $\mathbb{T}_1 = [(o : T\ (\ell\ viz)\ E)]$ and $Q' = (\mathsf{ret} : T\ (\ell\ viz))$, so the result follows trivially.
  - Subcase $viz = \mathbf{local}$:
    Impossible ($\mathbf{fun}(\mathbb{T})$ implies that last column must have visibility $\mathbf{output}$).
- Case:
  (APPLY ARG DET) (for T in **det**-space)
  $$\frac{((\mathbb{T}'\{^{e}/_{c}\}) \wedge \ell) \vdash R' \rightsquigarrow_o \mathbb{T}_1 \qquad c \notin \mathrm{fv}(R') \qquad \mathrm{dom}(\mathbb{T} \wedge \ell) \cap \mathrm{fv}(e) = \varnothing}{(c : T\ ((\ell' \wedge \ell)\ \mathbf{input})\ \varepsilon) :: (\mathbb{T}' \wedge \ell) \vdash [c = e] :: R' \rightsquigarrow_o \mathbb{T}_1}$$
  (TABLE INPUT)
  $$\frac{\Gamma, c :^{\ell' \wedge \ell} T \vdash^{\ell} \mathbb{T}' : Q_0}{\Gamma \vdash^{\ell} (c : T\ \ell'\ \mathbf{input}\ \varepsilon) :: \mathbb{T}' : (c : T\ (\ell' \wedge \ell)\ \mathbf{input}) :: Q_0}$$
  (ARG INPUT DET) (for T in **det**-space)
  $$\frac{\Gamma \vdash^{\ell' \wedge \ell} e : T \qquad \Gamma \vdash^{\ell} R' : Q_0\{^{e}/_{c}\} \to Q'}{\Gamma \vdash^{\ell} (c = e) :: R' : (c : T\ (\ell'\ \mathbf{input})) :: Q_0 \to Q'}$$
  We need: $\Gamma \vdash \mathbb{T}_1 : Q'[o/\mathsf{ret}]$.
  To use the induction hypothesis, we need to show that the following statements hold:
  - $(\mathbb{T}'\{^{e}/_{c}\}) \wedge \ell \vdash R' \rightsquigarrow_o \mathbb{T}_1$
    Given by the first assumption of (APPLY ARG DET).
  - $\Gamma \vdash^{\ell} \mathbb{T}'\{^{e}/_{c}\} : Q_0\{^{e}/_{c}\}$
    Follows by applying lemma 31 to the assumption of (TABLE INPUT) and the first assumption of (ARG INPUT DET).
  - $\Gamma \vdash^{\ell} R' : Q_0\{^{e}/_{c}\} \to Q'$
    Given by second assumption of (ARG INPUT DET).

- $\mathrm{dom}(\mathbb{T}'\{^e/_c\}) \cap \mathrm{dom}(\Gamma) = \varnothing$
  Follows directly from the assumption $\mathrm{dom}((c : T\ \ell'\ \textsf{input}\ \varepsilon) :: \mathbb{T}') \cap \mathrm{dom}(\Gamma) = \varnothing$.
- $\textsf{fun}(\mathbb{T}'\{^e/_c\})$:
  Follows trivially from the assumption $\textsf{fun}((c : T\ \ell'\ \textsf{input}\ \varepsilon) :: \mathbb{T}')$
- Last column of $\mathbb{T}'$ is at level ℓ:
  Follows from the same property of $(c : T\ \ell'\ \textsf{input}\ \varepsilon) :: \mathbb{T}'$

Thus, by induction hypothesis, we get $\Gamma \vdash^{\ell} \mathbb{T}_1 : Q'[o/\mathsf{ret}]$, as required.

– Case:
(Apply Arg Rnd Qry) (for T in **rnd** or **qry**-space)

$$\frac{\mathbb{T}' \wedge \ell \vdash R \rightsquigarrow_o \mathbb{T}_1' \qquad c \notin \mathrm{fv}(R) \qquad \mathrm{dom}(\mathbb{T}') \cap \mathrm{fv}(e) = \varnothing}{(c : T\ ((\ell' \wedge \ell)\ \textsf{input})\ \varepsilon) :: (\mathbb{T}' \wedge \ell) \vdash [c = e] :: R \rightsquigarrow_o (c : T\ ((\ell' \wedge \ell)\ \textsf{local})\ e) :: \mathbb{T}_1'}$$

(Table Input)

$$\frac{\Gamma, c :^{\ell' \wedge \ell} T \vdash^{\ell} \mathbb{T}' : Q_0}{\Gamma \vdash^{\ell} (c : T\ \ell'\ \textsf{input}\ \varepsilon) :: \mathbb{T}' : (c : T\ (\ell' \wedge \ell)\ \textsf{input}) :: Q_0}$$

(Arg Input Rnd Qry) (for T in **rnd** or **qry**-space)

$$\frac{\Gamma \vdash^{\ell' \wedge \ell} e : T \qquad \Gamma \vdash^{\ell} R' : Q_0 \to Q'}{\Gamma \vdash^{\ell} (c = e) :: R' : (c : T\ (\ell'\ \textsf{input})) :: Q_0 \to Q'}$$

We need: $\Gamma, c :^{\ell' \wedge \ell} T \vdash (c : T\ ((\ell' \wedge \ell)\ \textsf{local})\ e) :: \mathbb{T}_1' : Q'[o/\mathsf{ret}]$.
To use the induction hypothesis, we need to show that the following statements hold:

- $\mathbb{T}' \wedge \ell \vdash R' \rightsquigarrow_o \mathbb{T}_1$
  Given by the first assumption of (Apply Arg Rnd Qry).
- $\Gamma, c :^{\ell' \wedge \ell} T \vdash^{\ell} \mathbb{T}' : Q_0$
  Given by the assumption of (Table Input).
- $\Gamma, c :^{\ell' \wedge \ell} T \vdash^{\ell} R' : Q_0 \to Q'$
  Follows from applying lemma 30 to the second assumption of (Arg Input Rnd Qry) ($c \notin \Gamma$ follows from well-formedness of the environment in the first assumption of (Table Input).
- $\mathrm{dom}(\mathbb{T}') \cap \mathrm{dom}(\Gamma) = \varnothing$
  Follows directly from the assumption $\mathrm{dom}((c : T\ \ell'\ \textsf{input}\ \varepsilon) :: \mathbb{T}') \cap \mathrm{dom}(\Gamma) = \varnothing$.
- $\textsf{fun}(\mathbb{T})$:
  Follows trivially from the assumption $\textsf{fun}((c : T\ \ell'\ \textsf{input}\ \varepsilon) :: \mathbb{T}')$
- Last column of $\mathbb{T}'$ is at level ℓ:
  Follows from the same property of $(c : T\ \ell'\ \textsf{input}\ \varepsilon) :: \mathbb{T}'$

Thus, by induction hypothesis, we get $\Gamma, c :^{\ell' \wedge \ell} T \vdash^{\ell} \mathbb{T}_1 : Q'[o/\mathsf{ret}]$, which gives $\Gamma \vdash^{\ell} (c : T\ ((\ell' \wedge \ell)\ \textsf{local})\ E) :: \mathbb{T}_1' : Q'[o/\mathsf{ret}]$ by (Table Input)

– Case (Apply Expr):
(Apply Expr)

$$\frac{(\mathbb{T}' \wedge \ell) \vdash R \rightsquigarrow_o \mathbb{T}_1' \qquad c \notin \mathrm{fv}(R)}{(c : T\ (\ell\ viz)\ E) :: (\mathbb{T}' \wedge \ell) \vdash R \rightsquigarrow_o (c : T\ ((\ell \wedge \ell' viz)\ E) :: \mathbb{T}_1'}$$

- Subcase $viz = \textsf{output}$:
  (TABLE OUTPUT) (for Core tables)
  $$\frac{\Gamma \vdash^{\ell' \wedge \ell} E : T \qquad \Gamma, c :^{\ell' \wedge \ell} T \vdash^{\ell} \mathbb{T}' : Q_0}{\Gamma \vdash^{\ell} (c : T\ \ell'\ \textsf{output}\ E) :: \mathbb{T}' : ((c : T\ (\ell' \wedge \ell)\ \textsf{output}) :: Q_0)}$$
  (ARG OUTPUT)
  $$\frac{\Gamma \vdash T \qquad \Gamma \vdash^{\ell} R : Q_0 \to Q_0'}{\Gamma \vdash^{\ell} R : ((c : T\ \ell'\ \textsf{output}) :: Q_0) \to ((c : T\ (\ell' \wedge \ell)\ \textsf{output}) :: Q_0')}$$
  To apply the induction hypothesis, we need to show:
  * $(\mathbb{T}' \wedge \ell) \vdash R \rightsquigarrow_o \mathbb{T}_1'$
    Given by the first assumption of (APPLY EXPR).
  * $\Gamma, c :^{\ell' \wedge \ell} T \vdash^{\ell} \mathbb{T}' : Q_0$
    Given by the second assumption of (TABLE OUTPUT).
  * $\Gamma, c :^{\ell' \wedge \ell} T \vdash^{\ell} R : Q_0 \to Q_0'$
    Follows by applying lemma 30 to $\Gamma \vdash^{\ell} R : Q_0 \to Q_0'$.
  * $\mathrm{dom}(\mathbb{T}') \cap \mathrm{dom}(\Gamma) = \varnothing$.
    Follows from the assumption $\mathrm{dom}((c : T\ \ell'\ \textsf{output}\ E) :: \mathbb{T}') \cap \mathrm{dom}(\Gamma) = \varnothing$.
  * $\textsf{fun}(\mathbb{T}')$
    Follows from $\textsf{fun}((c : T\ \ell'\ \textsf{output}\ E) :: \mathbb{T}')$.
  * Last column of $\mathbb{T}'$ is at level ℓ.
    Follows from the fact that the last column of $(c : T\ \ell'\ \textsf{output}\ E) :: \mathbb{T}'$ is at level ℓ (note that $\mathbb{T}'$ is not empty).

  Hence, by induction hypothesis, we obtain $\Gamma, c :^{\ell' \wedge \ell} T \vdash \mathbb{T}_1 : Q_0'[o/\mathsf{ret}]$, and so by (TABLE OUTPUT), $\Gamma \vdash^{\ell} (c : T\ (\ell' \wedge \ell)\ \textsf{output}\ E) :: \mathbb{T}' : ((c : T\ (\ell' \wedge \ell)\ \textsf{output}) :: Q_0'[o/\mathsf{ret}])$
- Subcase $viz = \textsf{local}$:
  (TABLE LOCAL)
  $$\frac{\Gamma \vdash^{\ell' \wedge \ell} E : T \qquad \Gamma, c :^{\ell' \wedge \ell} T \vdash^{\ell} \mathbb{T}' : Q_0 \quad c \notin \mathrm{fv}(Q_0)}{\Gamma \vdash^{\ell} (c : T\ \ell'\ \textsf{local}\ E) :: \mathbb{T}' : Q_0}$$
  First, we need to show:
  * $(\mathbb{T}' \wedge \ell) \vdash R \rightsquigarrow_o \mathbb{T}_1'$
    Given by the first assumption of (APPLY EXPR).
  * $\Gamma, c :^{\ell' \wedge \ell} T \vdash^{\ell} \mathbb{T}' : Q_0$
    Given by the second assumption of (TABLE LOCAL).
  * $\Gamma, c :^{\ell' \wedge \ell} T \vdash^{\ell} R : Q_0 \to Q_0'$
    Follows vy applying lemma 30 to the assumption $\Gamma \vdash^{\ell} R : Q_0 \to Q_0'$.
  * $\mathrm{dom}(\mathbb{T}') \cap \mathrm{dom}(\Gamma) = \varnothing$.
    Follows from the assumption $\mathrm{dom}((c : T\ \ell'\ \textsf{local}\ E) :: \mathbb{T}') \cap \mathrm{dom}(\Gamma) = \varnothing$.
  * $\textsf{fun}(\mathbb{T}')$
    Follows from $\textsf{fun}((c : T\ \ell'\ \textsf{local}\ E) :: \mathbb{T}')$.
  * Last column of $\mathbb{T}'$ is at level ℓ.
    Follows from the fact that the last column of $(c : T\ \ell'\ \textsf{local}\ E) :: \mathbb{T}'$ is at level ℓ (note that $\mathbb{T}'$ is not empty).

Hence, by induction hypothesis, we obtain $\Gamma, c :^{\ell' \wedge \ell} T \vdash \mathbb{T}_1 : Q_0'[o/\mathsf{ret}]$, and so by (TABLE LOCAL),
$\Gamma \vdash^{\ell} (c : T\ (\ell' \wedge \ell)\ \textbf{local}\ E) :: \mathbb{T}' : Q_0'[o/\mathsf{ret}])$

■

Restatement of Proposition 1

(1) *If $\Gamma \vdash \mathbb{S} : Sty$ and $\mathbb{S} \to \mathbb{S}'$, then $\Gamma \vdash \mathbb{S}' : Sty$*
(2) *If $\Gamma \vdash^{pc} \mathbb{T} : Q$ and $\mathbb{T} \to \mathbb{T}'$ then $\Gamma \vdash^{pc} \mathbb{T}' : Q$.*
(3) *If $\Gamma \vdash^{pc} M : Q$ and $M \to M'$ then $\Gamma \vdash^{pc} M' : Q$.*

Proof: By simultaneous induction on the depth of derivations of $\mathbb{S} \to \mathbb{S}'$, $\mathbb{T} \to \mathbb{T}'$ and $M \to M'$.

- Case:
  (RED SCHEMA LEFT)
  $$\frac{\mathbb{T}_0 \to \mathbb{T}_0'}{(t = \mathbb{T}_0) :: \mathbb{S}_1 \to (t = \mathbb{T}_0') :: \mathbb{S}_1}$$
  Here, $\Gamma \vdash \mathbb{S}_1 : Sty$ must have been derived with (SCHEMA TABLE):
  (SCHEMA TABLE)
  $$\frac{\Gamma \vdash^{\mathsf{inst}} \mathbb{T}_0 : Q \qquad \Gamma, t : Q \vdash \mathbb{S}_1 : Sty'}{\Gamma \vdash (t = \mathbb{T}_0) :: \mathbb{S}_1 : (t : Q) :: Sty'}$$
  We have $\mathbb{T}_0 \to \mathbb{T}_0'$ and $\Gamma \vdash^{\mathsf{inst}} \mathbb{T}_0 : Q$, so $\Gamma \vdash^{\mathsf{inst}} \mathbb{T}_0' : Q$ by induction hypothesis. Thus, by (SCHEMA TABLE),
  $$\Gamma \vdash (t = \mathbb{T}_0') :: \mathbb{S}_1 : (t : Q) :: Sty'$$

- Case:
  (RED SCHEMA RIGHT)
  $$\frac{\mathbb{S}_1 \to \mathbb{S}_1' \qquad \mathsf{Core}(\mathbb{T}_0)}{(t = \mathbb{T}_0) :: \mathbb{S}_1 \to (t = \mathbb{T}_0) :: \mathbb{S}_1'}$$
  Again, $\Gamma \vdash \mathbb{S} : Sty$ must have bene derived with (SCHEMA TABLE):
  (SCHEMA TABLE)
  $$\frac{\Gamma \vdash^{\mathsf{inst}} \mathbb{T}_0 : Q \qquad \Gamma, t : Q \vdash \mathbb{S}_1 : Sty'}{\Gamma \vdash (t = \mathbb{T}_0) :: \mathbb{S}_1 : (t : Q) :: Sty'}$$
  By induction hypothesis, $\Gamma, t : Q \vdash \mathbb{S}_1' : Sty'$, so (SCHEMA TABLE) gives:
  $$\Gamma \vdash (t = \mathbb{T}_0) :: \mathbb{S}_1' : (t : Q) :: Sty'$$

- Case:
  (RED RIGHT STEP)
  $$\frac{\mathbb{T}_1 \to \mathbb{T}_1'}{\mathsf{col} :: \mathbb{T}_1 \to \mathsf{col} :: \mathbb{T}_1'}$$
  In this case, $\mathbb{T} = \mathsf{col} :: \mathbb{T}_1$, and we need to split on col:

- Subcase $\mathsf{col} = (c : T\ (\ell\ \textbf{input}\ \varepsilon))$:
  (TABLE INPUT)
  $$\frac{\Gamma, c :^{\ell \wedge pc} T \vdash^{pc} \mathbb{T}_1 : Q_1}{\Gamma \vdash^{pc} (c : T\ \ell\ \textbf{input}\ \varepsilon) :: \mathbb{T}_1 : (c : T\ \ell\ \textbf{input}) :: Q_1}$$
  Applying the induction hypothesis to the assumption of (RED RIGHT STEP) and (TABLE INPUT) gives:
  $$\Gamma, c :^{\ell \wedge pc} T \vdash \mathbb{T}_1 : Q_1$$
  Thus, by (TABLE INPUT), we obtain directly:
  $$\Gamma \vdash^{pc} (c : T\ (\ell\ \textbf{input}\varepsilon)) :: \mathbb{T}_1' : (c : T\ \ell\ \textbf{input}) :: Q_1$$
  as required.
- Subcase $\mathsf{col} = (c : T\ (\ell\ \textbf{local}\ M))$:
  (TABLE LOCAL)
  $$\frac{\Gamma \vdash^{\ell \wedge pc} M : Q_c @[(\mathsf{ret} : T\ \ell\ \textbf{output})] \quad \Gamma, \gamma(Q_c), c :^{\ell \wedge pc} T \vdash^{pc} \mathbb{T}_1 : Q_1 \quad c \notin \mathrm{fv}(Q_1)}{\Gamma \vdash^{pc} (c : T\ \ell\ \textbf{local}\ M) :: \mathbb{T}_1 : Q_1}$$
  Applying the induction hypothesis to $\mathbb{T}_1 \to \mathbb{T}_1'$ and the second assumption of (TABLE LOCAL) gives:
  $$\Gamma, \gamma(Q_c), c :^{\ell \wedge pc} T \vdash^{pc} \mathbb{T}_1' : Q_1$$
  Hence, by using (TABLE LOCAL) again, we obtain:
  $$\Gamma \vdash^{pc} (c : T\ \ell\ \textbf{local}\ M) :: \mathbb{T}_1' : Q_1$$
- Subcase $\mathsf{col} = (c : T\ (\ell\ \textbf{output}\ M))$:
  (TABLE OUTPUT)
  $$\frac{\Gamma \vdash^{\ell \wedge pc} M : Q_c @[(\mathsf{ret} : T\ \ell\ \textbf{output})] \quad \Gamma, \gamma(Q_c), c :^{\ell \wedge pc} T \vdash^{pc} \mathbb{T}_1 : Q_1}{\Gamma \vdash^{pc} (c : T\ \ell\ \textbf{output}\ M) :: \mathbb{T}_1 : Q_c @((c : T\ (\ell \wedge pc)\ \textbf{output}) :: Q_1)}$$
  Applying the induction hypothesis to $\mathbb{T}_1 \to \mathbb{T}_1'$ and the second assumption of (TABLE OUTPUT) gives:
  $$\Gamma, \gamma(Q_c), c :^{\ell \wedge pc} T \vdash^{pc} \mathbb{T}_1' : Q_1$$
  Thus, (TABLE OUTPUT) gives:
  $$\Gamma \vdash^{pc} (c : T\ \ell\ \textbf{output}\ M) :: \mathbb{T}_1' : Q_c @((c : T\ (\ell \wedge pc)\ \textbf{output}) :: Q_1)$$

– Case:
  (RED MODEL)
  $$\frac{M \to M'}{((c : T\ A\ M)) :: \mathbb{T}_1 \to ((c : T\ A\ M')) :: \mathbb{T}_1}$$
  We need to split on A:

- Subcase $A = \ell$ **local**:
  (TABLE LOCAL)
  $$\frac{\Gamma \vdash^{\ell \wedge pc} M : Q_c @[(\mathsf{ret} : T\ (\ell \wedge pc)\ \textbf{output})] \quad \Gamma, \gamma(Q_c), c :^{\ell \wedge pc} T \vdash^{pc} \mathbb{T}_1 : Q_1 \quad c \notin \mathrm{fv}(Q_1)}{\Gamma \vdash^{pc} (c : T\ \ell\ \textbf{local}\ M) :: \mathbb{T}_1 : Q_1}$$
  Applying the induction hypothesis to $M \to M'$ and the first assumption of (TABLE LOCAL) gives:
  $$\Gamma \vdash^{\ell \wedge pc} M' : Q_c @[(\mathsf{ret} : T\ (\ell \wedge pc)\ \textbf{output})]$$
  Thus, by (TABLE LOCAL):
  $$\Gamma \vdash^{pc} (c : T\ \ell\ \textbf{local}\ M') :: \mathbb{T}_1 : Q_1$$
- Subcase $A = \ell$ **output**:
  (TABLE OUTPUT)
  $$\frac{\Gamma \vdash^{\ell \wedge pc} M : Q_c @[(\mathsf{ret} : T\ \ell\ \textbf{output})] \quad \Gamma, \gamma(Q_c), c :^{\ell \wedge pc} T \vdash^{pc} \mathbb{T}_1 : Q_1}{\Gamma \vdash^{pc} (c : T\ \ell\ \textbf{output}\ M) :: \mathbb{T}_1 : Q_c @((c : T\ (\ell \wedge pc)\ \textbf{output}) :: Q_1)}$$
  By applying the induction hypothesis to $M \to M'$ and the first assumption of (TABLE OUTPUT), we get:
  $$\Gamma \vdash^{\ell \wedge pc} M' : Q_c @[(\mathsf{ret} : T\ (\ell \wedge pc)\ \textbf{output})]$$
  Hence, by (TABLE OUTPUT):
  $$\Gamma \vdash^{pc} (c : T\ \ell\ \textbf{output}\ M) :: \mathbb{T}_1 : Q_c @((c : T\ (\ell \wedge pc)\ \textbf{output}) :: Q_1)$$

– Case:
  (EQ APPL VARIABLE) (for Core($\mathbb{T}_f$))
  $$\frac{((\mathbb{T}_f \wedge \ell) \wedge viz) \vdash R \rightsquigarrow_o \mathbb{T}_1 \quad (\mathrm{locals}(\mathbb{T}_f) \cup \mathrm{inputs}(\mathbb{T}_f)) \cap \mathrm{fv}(\mathbb{T}_2) = \varnothing}{(o : T\ \ell\ viz\ (\mathbb{T}_f\ R)) :: \mathbb{T}_2 \to \mathbb{T}_1 @ \mathbb{T}_2}$$
  Split on *viz*:
  - Subcase $viz = \textbf{local}$
    Here, the (TABLE LOCAL) rule takes the form:
    (TABLE LOCAL)
    $$\frac{\Gamma \vdash^{\ell \wedge pc} (\mathbb{T}_f\ R) : Q_c @[(\mathsf{ret} : T\ (\ell \wedge pc)\ \textbf{output})] \quad \Gamma, \gamma(Q_c), o :^{\ell \wedge pc} T \vdash^{pc} \mathbb{T}_2 : Q_2 \quad o \notin \mathrm{fv}(Q_2)}{\Gamma \vdash^{pc} (o : T\ \ell\ \textbf{local}\ (\mathbb{T}_f\ R)) :: \mathbb{T}_2 : Q_2}$$
    We need: $\Gamma \vdash^{pc} \mathbb{T}_1 @ \mathbb{T}_2 : Q_2$.
    The first condition of (EQ APPL VARIABLE) gives $((\mathbb{T}_f \wedge \ell) \wedge \textbf{local}) \vdash R \rightsquigarrow_o \mathbb{T}_1$, and the first condition of (TABLE LOCAL) states that $\Gamma \vdash^{\ell \wedge pc} (\mathbb{T}_f\ R) : Q_c @[(\mathsf{ret} : T\ (\ell \wedge pc)\ \textbf{output})]$. Also, $\mathbb{T}_1$ here must be at level ℓ, which implies $\mathbb{T}_1 \wedge \ell = \mathbb{T}_1$. Thus, by lemma 36, we have $\gamma(Q_c @[(\mathsf{c} : T\ (\ell \wedge pc)\ \textbf{output})]) \leq \gamma(\mathbb{T}_1 \wedge \ell)$ and so $\Gamma, \gamma(Q_c @[(\mathsf{c} : T\ (\ell \wedge pc)\ \textbf{output})]) \leq \Gamma, \gamma(\mathbb{T}_1 \wedge \ell)$,

As the second condition of (TABLE LOCAL) gives $\Gamma, \gamma(Q_c @ [(\mathsf{c} : T\ (\ell \wedge pc)\ \textbf{output})]) \vdash^{pc} \mathbb{T}_2 : Q_2$ and it follows from the second assumption of (EQ APPL VARIABLE) that $\mathrm{dom}(\gamma(\mathbb{T}_1)) \cap \mathrm{dom}(\mathbb{T}_2) = \varnothing$, lemma 28 gives $\Gamma, \gamma(\mathbb{T}_1 \wedge \ell) \vdash^{pc} \mathbb{T}_2 : Q_2$.
Now, since $((\mathbb{T}_f \wedge \ell) \wedge \textbf{local}) \vdash R \rightsquigarrow_o \mathbb{T}_1$ and $(\mathbb{T}_f\ R)$ is well-typed in Γ, lemma 34 yields $\Gamma \vdash^{pc} \mathbb{T}_1 : []$. Hence, the desired result $\Gamma \vdash^{pc} \mathbb{T}_1 @ \mathbb{T}_2 : Q_2$ follows by lemma **??**.

- Subcase $viz = \textbf{output}$
Here, (TABLE OUTPUT) and (EQ APPL VARIABLE) become:
(TABLE OUTPUT)
$$\frac{\Gamma \vdash^{\ell \wedge pc} (\mathbb{T}_f\ R) : Q_c @ [(\mathsf{ret} : T\ (\ell \wedge pc)\ \textbf{output})] \quad \Gamma, \gamma(Q_c), o :^{\ell \wedge pc} T \vdash^{pc} \mathbb{T}_2 : Q_1}{\Gamma \vdash^{pc} (o : T\ \ell\ \textbf{local}\ (\mathbb{T}_f\ R)) :: \mathbb{T}_2 : Q_c @ [(\mathsf{o} : T\ (\ell \wedge pc)\ \textbf{output})] :: Q_1}$$
(EQ APPL VARIABLE) (for $\mathrm{Core}(\mathbb{T}_f)$)
$$\frac{(\mathbb{T}_f \wedge \ell) \vdash R \rightsquigarrow_o \mathbb{T}_1 \quad (\mathrm{locals}(\mathbb{T}_f) \cup \mathrm{inputs}(\mathbb{T}_f)) \cap \mathrm{fv}(\mathbb{T}_2) = \varnothing}{(o : T\ \ell\ \textbf{output}\ (\mathbb{T}_f\ R)) :: \mathbb{T}_2 \to \mathbb{T}_1 @ \mathbb{T}_2}$$
Thus, by applying Lemnma 37 to the first conditions of (TABLE OUTPUT) and (EQ APPL VARIABLE), we obtain:
$$\Gamma \vdash^{pc} \mathbb{T}_1 : Q_c @ [(o : T\ (\ell \wedge pc)\ \textbf{output})]$$
Now, by similar reasoning as above, we get $\Gamma, \gamma(\mathbb{T}_1 \wedge \ell) \vdash^{pc} \mathbb{T}_2 : Q_1$.
Thus, applying lemma **??** to the above result and the second assumption of (TABLE OUTPUT) gives:
$$\Gamma \vdash^{pc} \mathbb{T}_1 @ \mathbb{T}_2 : Q_c @ ([(o : T\ (\ell \wedge pc)\ \textbf{output})] :: Q_1)$$
as required.

– Case:
(EQ INDEX) (where $\mathrm{fv}(e_{index}, e_{size}) \cap (\mathrm{dom}(\mathbb{T}_f)) = \varnothing$)
$$\frac{e_{index} \text{ and } e_{size} \text{ contain no random} \quad \mathrm{Core}(\mathbb{T}_f)}{(\mathbb{T}_f\ R)[e_{index} < e_{size}] \to (\mathbb{T}_f[e_{index} < e_{size}]\ R)}$$
Here, $\Gamma \vdash^{pc} (\mathbb{T}_f\ R)[e_{index} < e_{size}] : Q$ must have been derived with:
(MODEL INDEXED)
$$\frac{e_{index} \text{ and } e_{size} \text{ contain no random} \quad \Gamma \vdash^{pc} (\mathbb{T}_f\ R) : Q' \quad \mathrm{locals}(Q') \cap \mathrm{fv}(e_{size}) = \varnothing \quad \Gamma \vdash^{pc} e_{index} : \textbf{mod}(e_{size})\ !\ spc}{\Gamma \vdash^{pc} (\mathbb{T}_f\ R)[e_{index} < e_{size}] : Q'[e_{size}]}$$
In turn, the last rule applied in the derivation of $\Gamma \vdash^{pc} (\mathbb{T}_f\ R) : Q'$ must have been
(MODEL APPL)
$$\frac{\Gamma \vdash^{pc} \mathbb{T}_f : Q'' \quad \Gamma \vdash^{pc} R : Q'' \to Q' \quad \textbf{fun}(Q'') \quad \mathrm{dom}(\mathbb{T}_f) \cap \mathrm{dom}(\Gamma) = \varnothing}{\Gamma \vdash^{pc} \mathbb{T}_f\ R : Q'}$$

Now, by Lemma 35, we have $\Gamma \vdash^{pc} \mathbb{T}_f[e_{index} < e_{size}] : Q''[e_{size}]$ which, combined with the fact that $\Gamma \vdash^{pc} R : Q''[E_s ize] \to Q'[E_s ize]$\$ (as indexing does not affect argument types) gives $\Gamma \vdash^{pc} (\mathbb{T}_f[e_{index} < e_{size}]\ R) : Q'[e_{size}]$, by applying (MODEL INDEXED) again.

– Case:
(EQ INDEX INNER)

$$\frac{M \to M'}{M[e_{index} < e_{size}] \to M'[e_{index} < e_{size}]}$$

$\Gamma \vdash^{pc} M[e_{index} < e_{size}] : Q$ must have been derived with:
(MODEL INDEXED)

$$\frac{e_{index} \text{ and } e_{size} \text{ contain no random} \quad \Gamma \vdash^{pc} M : Q' \quad \text{locals}(Q') \cap \text{fv}(e_{size}) = \varnothing \quad \Gamma \vdash^{pc} e_{index} : \textbf{mod}(e_{size})\,!\,d}{\Gamma \vdash^{pc} M[e_{index} < e_{size}] : Q'[e_{size}]}$$

By induction hypothesis, $\Gamma \vdash^{pc} M : Q'$ Hence, by (MODEL INDEXED):

$$\Gamma \vdash^{pc} M'[e_{index} < e_{size}] : Q'[e_{size}]$$

■

F.3 Proof of Proposition 2

As usual, we need to state and prove some additional lemmas first:

Lemma 38. *If $\Gamma \vdash^{\ell} E : T$, then $\text{fv}(E) \subseteq \text{dom}(\Gamma)$ and $\text{fv}(T) \subseteq \text{dom}(\Gamma)$.*

Proof: By straightforward induction on the depth of derivation of $\Gamma \vdash^{\ell} E : T$. ■

Lemma 39. *If $\mathsf{Core}(\mathbb{T})$ and $\Gamma \vdash^{pc} \mathbb{T} : Q$ then $\text{dom}(\mathbb{T}) \cap \text{dom}(\Gamma) = \varnothing$.*

Proof: By straightforward induction on the depth of derivation of $\Gamma \vdash^{pc} \mathbb{T} : Q$. ■

Lemma 40. *If $\Gamma \vdash^{\ell} \mathbb{T}\ R : Q$, then all columns of Q have visibility* **output**.

Proof: The last rule in derivation of $\Gamma \vdash^{\ell} \mathbb{T}\ R : Q$ must be (MODEL APPL), whose second assumption is $\Gamma \vdash^{\ell} R : Q \to Q'$. The lemma can then be easily proved by induction on the depth of derivation of $\Gamma \vdash^{\ell} R : Q \to Q'$. ■

Lemma 41. *If $\text{Core}(\mathbb{T})$ $\Gamma \vdash^{\ell} \mathbb{T}\ R : Q'$, then for any o and $viz \neq$ **input**, there is $\mathbb{T}_1$ such that $((\mathbb{T} \wedge \ell) \wedge viz) \vdash R \rightsquigarrow_o \mathbb{T}_1$*

Proof: The judgment $\Gamma \vdash^{\ell} (\mathbb{T}\ R) : Q'$ must have been derived with (MODEL APPL):
(MODEL APPL)

$$\frac{\Gamma \vdash^{\ell} \mathbb{T} : Q \quad \textbf{fun}(Q) \quad \Gamma \vdash^{\ell} Q \to Q'}{\Gamma \vdash^{\ell} \mathbb{T}\ R : Q'}$$

Thus, the statement of the lemma can be reformulated as follows:

If Core($\mathbb{T}$) and $\Gamma \vdash^{\ell} \mathbb{T} : Q$ and **fun**(Q) and $\Gamma \vdash^{\ell} R : Q \to Q'$ then for any o and $viz \neq$ **input**, there is $\mathbb{T}_1$ such that $((\mathbb{T} \wedge \ell) \wedge viz) \vdash R \rightsquigarrow_o \mathbb{T}_1$

This version of the lemma can be proven by induction on the derivation of $\Gamma \vdash^{\ell} \mathbb{T} : Q$:

- Case:
  (TABLE INPUT) (version for Core tables)
  $$\frac{\Gamma, c :^{\ell'} T \vdash \mathbb{T}_0 : Q_0}{\Gamma \vdash (c : T\ \ell'\ \textbf{input}\ \varepsilon) :: \mathbb{T}_0 : (c : T\ \ell'\ \textbf{input}) :: Q_0}$$
  - Subcase space(T) = **det**:
    Here, $\Gamma \vdash^{\ell} R : Q \to Q'$ must have been derived using:
    (ARG INPUT DET) (for T in **det**-space)
    $$\frac{\Gamma \vdash^{\ell' \wedge \ell} e : T \qquad \Gamma \vdash^{\ell} R_0 : Q_0\{^{e}/_{c}\} \to Q'}{\Gamma \vdash^{\ell} ((c = e) :: R_0) : ((c : T\ \ell'\ \textbf{input}) :: Q_0) \to Q'}$$
    By lemma 31, $\Gamma \vdash \mathbb{T}_0\{^{e}/_{c}\} : Q_0\{^{e}/_{c}\}$. Thus, by the induction hypothesis, there is a $\mathbb{T}_2$ such that $((\mathbb{T} \wedge \ell) \wedge viz) \vdash R \rightsquigarrow_o \mathbb{T}_2$. By lemma 38, $\text{fv}(e) \subseteq \text{dom}(\Gamma)$, and by lemma 39, $\text{dom}(\Gamma) \cap \text{dom}(\mathbb{T}_0) = \varnothing$, which gives $\text{fv}(e) \cap \text{fv}(\mathbb{T}_0) = \varnothing$. Since $c \notin \text{fv}(\Gamma)$, we know that $c \notin R_0$. Hence, (APPLY ARG DET) gives
    $$(c : T\ ((\ell' \wedge \ell)\ \textbf{input})\ \varepsilon) :: ((\mathbb{T}_0 \wedge \ell) \wedge viz) \vdash [c = e] :: R_0 \rightsquigarrow_o \mathbb{T}_2$$
  - Subcase space(T) $\neq$ **det**:
    (ARG INPUT RND QRY) (for T in **rnd** or **qry**-space)
    $$\frac{\Gamma \vdash^{\ell' \wedge \ell} e : T \qquad \Gamma \vdash^{\ell} R_0 : Q_0 \to Q'}{\Gamma \vdash^{\ell} ((c = e) :: R_0) : ((c : T\ \ell'\ \textbf{input}) :: Q_0) \to Q'}$$
    By lemma 30, $\Gamma, c :^{\ell' \wedge \ell} T \vdash^{\ell} R_0 : Q_0 \to Q'$, so the induction hypothesis gives $((\mathbb{T}_0 \wedge \ell) \wedge Q) \vdash R_0 \rightsquigarrow_o \mathbb{T}_2$ for some $\mathbb{T}_2$.
    By lemma 38, $\text{fv}(e) \subseteq \text{dom}(\Gamma)$, and by lemma 39, $\text{dom}(\Gamma) \cap \text{dom}(\mathbb{T}_0) = \varnothing$, which gives $\text{fv}(e) \cap \text{fv}(\mathbb{T}_0) = \varnothing$.
    Since $c \notin \text{dom}(\Gamma)$, we also know that $c \notin \text{fv}(R_0)$.
    Therefore, by (APPLY ARG RND QRY), we have
    $$(c : T\ (\ell' \wedge \ell)\ \textbf{input}\ \varepsilon) :: ((\mathbb{T}_0 \wedge \ell) \wedge viz) \vdash (c = e) :: R_0 \rightsquigarrow_o (c : T\ (\ell' \wedge \ell)\ \textbf{input}\ E) :: \mathbb{T}_2$$
    as required.
- Case:
  (TABLE LOCAL) (version for Core tables)
  $$\frac{\Gamma \vdash^{\ell' \wedge \ell} E : T \quad \Gamma, c :^{\ell' \wedge \ell} T \vdash^{\ell} \mathbb{T}_0 : Q \quad c \notin \text{fv}(Q)}{\Gamma \vdash (c : T\ \ell'\ \textbf{local}\ E) :: \mathbb{T} : Q}$$
  Again, we have $\Gamma, c :^{\ell' \wedge \ell} \vdash^{\ell} R : Q \to Q'$ by lemma 30, so the induction hypothesis gives $((\mathbb{T}_0 \wedge \ell) \wedge viz) \vdash R_0 \rightsquigarrow_o \mathbb{T}_2$ for some $\mathbb{T}_2$.
  As $c \notin \text{fv}(R)$ follows from the same reasoning as in the previous case, the rule (APPLY EXPR) gives
  $$(c : T\ (\ell' \wedge \ell)\ \textbf{local}\ E) :: ((\mathbb{T}_0 \wedge \ell) \wedge viz) \vdash R \rightsquigarrow_o (c : T\ (\ell' \wedge \ell)\ \textbf{local}\ E) :: \mathbb{T}_2$$

– Case:
(TABLE OUTPUT) (version for Core tables)

$$\frac{\Gamma_1 \vdash^{\ell'} E : T \qquad \Gamma, c :^{\ell'} T \vdash \mathbb{T}_0 : Q_0}{\Gamma_1 \vdash (c : T\ \ell'\ \textbf{output}\ E) :: \mathbb{T}_0 : ((c : T\ \ell'\ \textbf{output}) :: Q_0)}$$

In this case, $\Gamma \vdash^{\ell} R : Q \to Q'$ must have been derived with
(ARG OUTPUT)

$$\frac{\Gamma \vdash T \qquad \Gamma \vdash^{\ell} R : Q_0 \to Q_0'}{\Gamma \vdash^{\ell} R : (c : T\ (\ell'\ \textbf{output})) :: Q_0 \to (c : T\ ((\ell' \wedge \ell)\ \textbf{output})) :: Q_0'}$$

Now, since the first column of $\mathbb{T}$ is at level **output**, it might also be the last column (i.e. $\mathbb{T}_0 = []$ and $c = \mathsf{ret}$). Thus, we need to consider two subcases:

- If $\mathbb{T}_0 = []$, then $\mathbb{T} = [(\mathsf{ret} : T\ \ell'\ \textbf{output}\ E)]$ and $(\mathbb{T} \wedge \ell) \wedge viz = [(\mathsf{ret} : T\ (\ell' \wedge \ell)\ viz\ E)]$. Thus, by (APPLY OUTPUT),

$$[(\mathsf{ret} : T\ (\ell' \wedge \ell)\ viz\ E)] \rightsquigarrow_o [(o : T\ (\ell' \wedge \ell)\ viz\ E)]$$

- If $\mathbb{T}_0 \neq []$, then $(\mathbb{T} \wedge \ell) \wedge viz = (c : T\ (\ell' \wedge \ell)\ viz\ E) :: ((\mathbb{T}_0 \wedge \ell) \wedge viz)$. By using lemma 30 again, we get $((\mathbb{T}_0 \wedge \ell) \wedge viz) \vdash R_0 \rightsquigarrow_o \mathbb{T}_2$ for some $\mathbb{T}_2$ by induction hypothesis. Thus, the induction hypothesis states that$((\mathbb{T}_0 \wedge \ell) \wedge viz) \vdash R \rightsquigarrow_o \mathbb{T}_2$ for some $\mathbb{T}_2$. As $c \notin \mathrm{fv}(R)$ follows from the same reasoning as in the previous cases, we have:

$$(c : T\ (\ell' \wedge \ell)\ viz\ E) :: ((\mathbb{T}_0 \wedge \ell) \wedge viz) \vdash R \rightsquigarrow_o (c : T\ (\ell' \wedge \ell)\ viz\ E) :: \mathbb{T}_2$$

■

Lemma 42. *If $\Gamma \vdash^{pc} M[e_{index} < e_{size}] : Q'$, then there is M' such that $M[e_{index} < e_{size}] \to M'$.*

Proof:
By induction on the depth of derivation of $\Gamma \vdash M[e_{index} < e_{size}] : Q$ (where $Q' = Q[e_{size}]$):
(MODEL INDEXED)

$$\frac{\begin{array}{l} e_{index} \text{ and } e_{size} \text{ contain no random} \\ \Gamma \vdash^{pc} M : Q \quad \mathrm{locals}(Q) \cap \mathrm{fv}(e_{size}) = \varnothing \\ \Gamma \vdash^{pc} e_{index} : \textbf{mod}(e_{size})\ !\ d \end{array}}{\Gamma \vdash^{pc} M[e_{index} < e_{size}] : Q[e_{size}]}$$

– Case $M = E$: Here, by (RED INDEX EXPR), we have:

$$E[e_{index} < e_{size}] \to E$$

– Case $M = \mathbb{T}\ R$: In this case, (EQ INDEX) gives:

$$(\mathbb{T}\ R)[e_{index} < e_{size}] \to (\mathbb{T}[e_{index} < e_{size}])\ R$$

- Case $M = M_0[E_1 < e_2]$:
  The second assumption of (MODEL INDEXED) gives:

  $$\Gamma \vdash^{pc} M_0[e_1 < e_2] : Q$$

  Thus, by induction hypothesis, there exists M'' such that:

  $$M_0[e_1 < e_2] \to M''$$

  Hence, by (EQ INDEX INNER):

  $$(M_0[e_1 < e_2])[e_{index} < e_{size}] \to M''[e_{index} < e_{size}]$$

■

Lemma 43. *If $\Gamma \vdash^{pc} \mathbb{T} : Q$ either* $\mathsf{Core}(\mathbb{T})$ *or there is $\mathbb{T}'$ such that $\mathbb{T} \to \mathbb{T}'$.*

Proof: By induction on the depth of derivation of $\Gamma \vdash^{pc} \mathbb{T} : Q$

- Case:
  (TABLE INPUT)
  $$\frac{\Gamma, c :^{\ell \wedge viz} T \vdash^{pc} \mathbb{T}_0 : Q_0}{\Gamma \vdash (c : T\ \ell\ \textbf{input}\ \varepsilon) :: \mathbb{T}_0 : (c : T\ (\ell \wedge viz)\ \textbf{input}) :: Q_0}$$
  By induction hypothesis, either $\mathsf{Core}(\mathbb{T}_0)$ or there exists a $\mathbb{T}_0'$ such that $\mathbb{T}_0 \to \mathbb{T}_0'$.
  If $\mathsf{Core}(\mathbb{T}_0)$, then $\mathsf{Core}((c : T\ \ell\ \textbf{input}\ \varepsilon) :: \mathbb{T}_0)$, by definition of Core.
  If $\mathbb{T}_0 \to \mathbb{T}_0'$ for some $\mathbb{T}_0'$, then, since $\mathsf{Core}((c : T\ \ell\ \textbf{input}\ \varepsilon))$ by the definition of Core, (RED RIGHT STEP) gives

  $$(c : T\ \ell\ \textbf{input}\ \varepsilon) :: \mathbb{T}_0 \to (c : T\ \ell\ \textbf{input}\ \varepsilon) :: \mathbb{T}_0'$$

- Case:
  (TABLE LOCAL)
  $$\frac{\Gamma \vdash^{\ell \wedge pc} M : Q_c @[(\mathsf{ret} : T\ \ell\ \textbf{output})] \quad \Gamma, \gamma(Q_c), c :^{\ell \wedge pc} T \vdash^{pc} \mathbb{T}_0 : Q_0 \quad c \notin \mathrm{fv}(Q_0)}{\Gamma \vdash^{pc} (c : T\ \ell\ \textbf{local}\ M) :: \mathbb{T}_0 : Q_0}$$
  By induction hypothesis, either $\mathsf{Core}(\mathbb{T}_0)$ or there exists a $\mathbb{T}_0'$ such that $\mathbb{T}_0 \to \mathbb{T}_0'$.
  - If $\mathsf{Core}(\mathbb{T}_0)$, and $M = E$ then $\mathsf{Core}((c : T\ \ell\ \textbf{local}\ M) :: \mathbb{T}_0)$, by definition of Core.
  - If $M = E$ and $\mathbb{T}_0 \to \mathbb{T}_0'$ for some $\mathbb{T}_0'$, then, as $\mathsf{Core}((c : T\ \ell\ \textbf{local}\ E))$, by (RED RIGHT STEP) we get

    $$(c : T\ \ell\ \textbf{local}\ E) :: \mathbb{T}_0 \to (c : T\ \ell\ \textbf{local}\ E) :: \mathbb{T}_0'$$

  - If $M = \mathbb{T}_f\ R$ then by Lemma 41, $((\mathbb{T}_f \wedge \ell) \wedge \textbf{local}) \vdash R \rightsquigarrow_c \mathbb{T}_1$ for some $\mathbb{T}_1$. Also, Lemma 40 states that all columns of Q_c are at level **output**. As the derivation of $\Gamma \vdash^{\ell} (\mathbb{T}_f\ R) : Q_c @[(\mathsf{ret} : T\ \ell\ \textbf{output})]$ requires $\Gamma \vdash \mathbb{T}_f : Q_f$ for some Q_f, we have $\mathrm{dom}(\mathbb{T}_f) \cap \mathrm{dom}(\Gamma) = \varnothing$ by Lemma 39, which implies $(\mathrm{locals}(\mathbb{T}_f) \cup \mathrm{inputs}(\mathbb{T}_f)) \cap \mathrm{dom}(\Gamma, \gamma(Q_c), c :^{\ell} T) = \varnothing$. Hence, by Part 2 of Lemma 22, we have $(\mathrm{locals}(\mathbb{T}_f) \cup \mathrm{inputs}(\mathbb{T}_f)) \cap \mathrm{fv}(\mathbb{T}_0) = \varnothing$.
    Thus, (EQ APPL VARIABLE) gives

    $$(c : T\ \ell\ \textbf{local}\ (\mathbb{T}_f\ R)) :: \mathbb{T}_0 \to \mathbb{T}_1 @ \mathbb{T}_0$$

- If $M = M_0[e_{index} < e_{size}]$, then, by Lemma 42, there exists M' such that $M_0[e_{index} < e_{size}] \to M'$, and so, by (RED MODEL),

$$(c : T\ \ell\ \textsf{local}\ M_0[e_{index} < e_{size}]) :: \mathbb{T}_0 \to (c : T\ \ell\ \textsf{local}\ M') :: \mathbb{T}_0$$

– Case:
(TABLE OUTPUT)
$\Gamma \vdash^{\ell \wedge pc} M : Q_c@[(\mathsf{ret} : T\ (\ell \wedge pc)\ \textsf{output})]$
$\Gamma, \gamma(Q_c), c :^{\ell \wedge pc} T \vdash^{pc} \mathbb{T}_0 : Q_0$
$\overline{}$
$\Gamma \vdash^{pc} (c : T\ \ell\ \textsf{output}\ M) :: \mathbb{T}_0 : Q_c@((c : T\ (\ell \wedge pc)\ \textsf{output}) :: Q_0)$
By induction hypothesis, either $\mathsf{Core}(\mathbb{T}_0)$ or $\mathbb{T}_0 \to \mathbb{T}_0'$ for some $\mathbb{T}_0'$.

- If $\mathsf{Core}(\mathbb{T}_0)$, and $M = E$ then $\mathsf{Core}((c : T\ \ell\ \textsf{output}\ M) :: \mathbb{T}_0)$.
- If $M = E$ and $\mathbb{T}_0 \to \mathbb{T}_0'$ for some $\mathbb{T}_0'$, then $\mathsf{Core}((c : T\ \ell\ \textsf{output}\ E))$, and so (RED RIGHT STEP) gives:

$$(c : T\ \ell\ \textsf{output}\ E) :: \mathbb{T}_0 \to (c : T\ \ell\ \textsf{output}\ E) :: \mathbb{T}_0'$$

- If $M = \mathbb{T}_f\ R$ then Lemma 41 says that $\mathbb{T}_f \wedge \ell \vdash R \rightsquigarrow_c \mathbb{T}_1$ for some $\mathbb{T}_1$. Also, Lemma 40 states that all columns of Q_c are at level **output**. As the derivation of $\Gamma \vdash^{\ell} (\mathbb{T}_f\ R) : Q_c@[(\mathsf{ret} : T\ \ell\ \textsf{output})]$ requires $\Gamma \vdash \mathbb{T}_f : Q_f$ for some Q_f, we have $\mathrm{dom}(\mathbb{T}_f) \cap \mathrm{dom}(\Gamma) = \varnothing$ by Lemma 39, which implies $(\mathrm{locals}(\mathbb{T}_f) \cup \mathrm{inputs}(\mathbb{T}_f)) \cap \mathrm{dom}(\Gamma, \gamma(Q_c), c :^{\ell} T) = \varnothing$. Hence, by Part 2 of Lemma 22, we have $(\mathrm{locals}(\mathbb{T}_f) \cup \mathrm{inputs}(\mathbb{T}_f)) \cap \mathrm{fv}(\mathbb{T}_0) = \varnothing$.
Thus, (EQ APPL VARIABLE) gives

$$(c : T\ \ell\ \textsf{output}\ (\mathbb{T}_f\ R)) :: \mathbb{T}_0 \to \mathbb{T}_1@\mathbb{T}_0$$

- If $M = M_0[e_{index} < e_{size}]$, then, by Lemma 42, there exists M' such that $M_0[e_{index} < e_{size}] \to M'$, and so, by (RED MODEL),

$$(c : T\ \ell\ \textsf{output}\ M_0[e_{index} < e_{size}]) :: \mathbb{T}_0 \to (c : T\ \ell\ \textsf{output}\ M') :: \mathbb{T}_0$$

– Case:
(TABLE EMPTY)
$\Gamma \vdash \diamond$
$\overline{}$
$\Gamma \vdash [] : []$
Obviously, $\mathsf{Core}([])$ (this follows from the definition of Core for a table with $n = 0$ columns).

■

Restatement of Proposition 2 *If $\Gamma \vdash^{pc} \mathbb{S} : Sty$ either* $\mathsf{Core}(\mathbb{S})$ *or there is $\mathbb{S}'$ such that $\mathbb{S} \to \mathbb{S}'$.*

Proof: By induction on the derivation of $\Gamma \vdash^{pc} \mathbb{S} : Q$.

– Case:
(Schema [])
$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash [] : []}$$
Here, obviously, Core([]).

– Case:
(Schema Table)
$$\frac{\Gamma \vdash^{\textbf{inst}} \mathbb{T} : Q \quad \textbf{table}(Q) \quad \Gamma, t : Q \vdash \mathbb{S}_0 : Sty}{\Gamma \vdash (t = \mathbb{T}) :: \mathbb{S}_0 : (t : Q) :: Sty}$$
By lemma 43, either Core($\mathbb{T}$) or there exists a $\mathbb{T}'$ such that $\mathbb{T} \to \mathbb{T}'$. By induction hypothesis, either either Core($\mathbb{S}$) or there exists a $\mathbb{S}'$ such that $\mathbb{S} \to \mathbb{S}'$.
If Core($\mathbb{T}$), then :
- If Core($\mathbb{S}_0$), then Core($(t = \mathbb{T}) :: \mathbb{S}_0$)
- If $\mathbb{S}_0 \to \mathbb{S}_0'$, for some $\mathbb{S}'$, then by (Red Schema Right), $(t = \mathbb{T}) :: \mathbb{S}_0 \to (t = \mathbb{T}) :: \mathbb{S}_0'$

If $\mathbb{T} \to \mathbb{T}'$, then, by (Red Schema Left), $(t = \mathbb{T}) :: \mathbb{S}_0 \to (t = \mathbb{T}') :: \mathbb{S}_0$

■

F.4 Proof of Proposition 3

To prove termination of reduction, we construct a non-negative measure that is reduced by the reduction relation $\mathbb{S} \to \mathbb{S}'$. The measure counts the function applications and model expressions.

Measure on $\mathbb{S}$ reduced by $\to$: $m(\mathbb{S})$, $m(\mathbb{T})$, $m(M)$

$m([]) = 0$
$m((t = \mathbb{T}) :: \mathbb{S}) = m(\mathbb{T}) + m(\mathbb{S})$

$m([]) = 0$
$m((c : T\ (\ell\ viz)\ M) :: \mathbb{T}) = m(M) + m(\mathbb{T})$

$m(\mathbb{T}\ R) = 1$
$m(M[e_{index} < e_{size}]) = 1 + m(M)$
$m(E) = 0$

Now we need to show that this measure is decreasing, by proving the following lemma:

Lemma 44. (1) *If $M \to M'$ then $m(M') < m(M)$*
(2) *If $\mathbb{T} \to \mathbb{T}'$ then $m(\mathbb{T}') < m(\mathbb{T})$*
(3) *If $\mathbb{S} \to \mathbb{S}'$ then $m(\mathbb{S}') < m(\mathbb{S})$*

Proof:

(1) By induction on the depth of derivation of $M \to M'$

- Case:
  (RED INDEX) (where $\mathrm{fv}(e_{index}, e_{size}) \cap (\mathrm{dom}(\mathbb{T})) = \varnothing$)
  $$\frac{e_{index} \text{ and } e_{size} \text{ contain no random} \quad \mathrm{Core}(\mathbb{T})}{(\mathbb{T}\ R)[e_{index} < e_{size}] \to (\mathbb{T}[e_{index} < e_{size}]\ R)}$$
  $m(RHS) = 1 < 2 = m(LHS)$, as required.
- Case:
  (RED INDEX EXPR)
  $$\frac{}{E[e_{index} < e_{size}] \to E}$$
  $m(RHS) = 0 < 1 = m(LHS)$, as required.
- Case:
  (RED INDEX INNER)
  $$\frac{M_1 \to M_1'}{M_1[e_{index} < e_{size}] \to M_1'[e_{index} < e_{size}]}$$
  By induction hypothesis, $m(M_1') < m(M_1)$. Thus, $m(RHS) = 1 + m(M_1') < 1 + m(M_1) = m(LHS)$, as required.

(2) By induction on the depth of derivation of $\mathbb{T} \to \mathbb{T}'$

- Case:
  (RED APPL VARIABLE) (for $\mathrm{Core}(\mathbb{T})$)
  $$\frac{((\mathbb{T}_0 \wedge \ell) \wedge viz) \vdash R \rightsquigarrow_o \mathbb{T}_1 \qquad (\mathrm{locals}(\mathbb{T}_0) \cup \mathrm{inputs}(\mathbb{T}_0)) \cap (\mathrm{fv}(\mathbb{T}_0') \cup \mathrm{dom}(\mathbb{T}_0')) = \varnothing}{(o : T'\ \ell\ viz\ (\mathbb{T}_0\ R)) :: \mathbb{T}_0' \to \mathbb{T}_1 @ \mathbb{T}_0'}$$
  Since $\mathrm{Core}(\mathbb{T}_1)$ by the definition of $\rightsquigarrow$, we have $m(\mathbb{T}_1) = 0$ and so $m(\mathbb{T}_1 @ \mathbb{T}_0') = m(\mathbb{T}_0')$.
  Thus, $m(RHS) = m(\mathbb{T}_0') < 1 + m(\mathbb{T}_0') = m(LHS)$, as required.
- Case:
  (RED MODEL)
  $$\frac{M \to M'}{(c : T\ \ell\ viz\ M) :: \mathbb{T}_1 \to (c : T\ \ell\ viz\ M') :: \mathbb{T}_1}$$
  Part 1 of the lemma gives $m(M') < m(M)$. Thus, $m(RHS) = m(M') + m(\mathbb{T}_1) < m(M) + m(\mathbb{T}_1) = m(LHS)$, as required.
- Case:
  (RED TABLE RIGHT)
  $$\frac{\mathbb{T}_1 \to \mathbb{T}_1' \quad \mathsf{Core}(\mathsf{col})}{\mathsf{col} :: \mathbb{T}_1 \to \mathsf{col} :: \mathbb{T}_1'}$$
  By induction hypothesis, $m(\mathbb{T}_1') < m(\mathbb{T}_1)$. Also, by definition of Core, we have $m(\mathsf{col}) = 0$.
  Hence, $m(RHS) = m(\mathbb{T}_1') < m(\mathbb{T}_1) = m(LHS)$, as required.

(3) By induction on the depth of derivation of $\mathbb{S} \to \mathbb{S}'$

- Case:
  (RED SCHEMA LEFT)
  $$\frac{\mathbb{T} \to \mathbb{T}'}{(t = \mathbb{T}) :: \mathbb{S}_1 \to (t = \mathbb{T}') :: \mathbb{S}_1}$$
  By part 2, we have $m(\mathbb{T}') < m(\mathbb{T})$.
  Hence, $m(RHS) = m(\mathbb{T}') + m(\mathbb{S}_1) < m(\mathbb{T}) + m(\mathbb{S}_1) = m(LHS)$, as required.

- Case:
  (RED SCHEMA RIGHT)
  $$\frac{\mathbb{S}_1 \rightarrow \mathbb{S}'_1 \quad \mathsf{Core}(\mathbb{T})}{(t = \mathbb{T}) :: \mathbb{S}_1 \rightarrow (t = \mathbb{T}) :: \mathbb{S}'_1}$$
  By induction hypothesis, we have $m(\mathbb{S}'_1) < m(\mathbb{S}_1)$. Also, $\mathsf{Core}(\mathbb{T})$ implies that $m(\mathbb{T}) = 0$.
  Thus, $m(RHS) = m(\mathbb{S}'_1) < m(\mathbb{S}_1) = m(LHS)$, as required.

■

Restatement of Proposition 3 *No infinite chain* $\mathbb{S}_0 \rightarrow \mathbb{S}_1 \rightarrow \dots$ *exists.*

Proof: Such a chain would contradict the fact that our non-negative measure is reduced by the reduction relation. ■