GenSQL: A Probabilistic Programming System for Querying Generative Models of Database Tables

MATHIEU HUOT, Massachusetts Institute of Technology, USA
MATIN GHAVAMI, Massachusetts Institute of Technology, USA
ALEXANDER K. LEW, Massachusetts Institute of Technology, USA
ULRICH SCHAECHTLE, Digital Garage, Japan
CAMERON E. FREER, Massachusetts Institute of Technology, USA
ZANE SHELBY, Digital Garage, Japan
MARTIN C. RINARD, Massachusetts Institute of Technology, USA
FERAS A. SAAD, Carnegie Mellon University, USA
VIKASH K. MANSINGHKA, Massachusetts Institute of Technology, USA

This article presents GenSQL, a probabilistic programming system for querying probabilistic generative models of database tables. By augmenting SQL with only a few key primitives for querying probabilistic models, GenSQL enables complex Bayesian inference workflows to be concisely implemented. GenSQL's query planner rests on a unified programmatic interface for interacting with probabilistic models of tabular data, which makes it possible to use models written in a variety of probabilistic programming languages that are tailored to specific workflows. Probabilistic models may be automatically learned via probabilistic program synthesis, hand-designed, or a combination of both. GenSQL is formalized using a novel type system and denotational semantics, which together enable us to establish proofs that precisely characterize its soundness guarantees. We evaluate our system on two case real-world studies—an anomaly detection in clinical trials and conditional synthetic data generation for a virtual wet lab—and show that GenSQL more accurately captures the complexity of the data as compared to common baselines. We also show that the declarative syntax in GenSQL is more concise and less error-prone as compared to several alternatives. Finally, GenSQL delivers a 1.7-6.8x speedup compared to its closest competitor on a representative benchmark set and runs in comparable time to hand-written code, in part due to its reusable optimizations and code specialization.

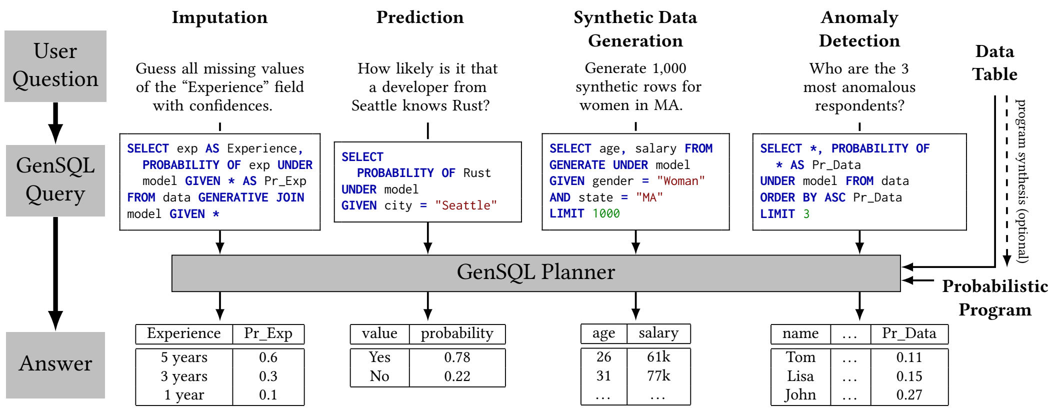
CCS Concepts: • **Mathematics of computing → Bayesian computation**; *Statistical software*; • **Software and its engineering → Semantics**.

Additional Key Words and Phrases: generative modeling, Bayesian data science, probabilistic query language

ACM Reference Format:
Mathieu Huot, Matin Ghavami, Alexander K. Lew, Ulrich Schaechtle, Cameron E. Freer, Zane Shelby, Martin C. Rinard, Feras A. Saad, and Vikash K. Mansinghka. 2024. GenSQL: A Probabilistic Programming System for Querying Generative Models of Database Tables. *Proc. ACM Program. Lang.* 8, PLDI, Article 179 (June 2024), 26 pages. https://doi.org/10.1145/3656409

Authors' addresses: Mathieu Huot, mhuot@mit.edu, Massachusetts Institute of Technology, Cambridge, MA, USA; Matin Ghavami, mghavami@mit.edu, Massachusetts Institute of Technology, Cambridge, MA, USA; Alexander K. Lew, alexlew@mit.edu, Massachusetts Institute of Technology, Cambridge, MA, USA; Ulrich Schaechtle, ulli-schaechtle@garage.co.jp, Digital Garage, Tokyo, Japan; Cameron E. Freer, freer@mit.edu, Massachusetts Institute of Technology, Cambridge, MA, USA; Zane Shelby, pldi@zane.io, Digital Garage, Tokyo, Japan; Martin C. Rinard, rinard@csail.mit.edu, Massachusetts Institute of Technology, Cambridge, MA, USA; Feras A. Saad, fsaad@cmu.edu, Carnegie Mellon University, Pittsburgh, PA, USA; Vikash K. Mansinghka, vkm@mit.edu, Massachusetts Institute of Technology, Cambridge, MA, USA.

ACM 2475-1421/2024/6-ART179
https://doi.org/10.1145/3656409


User Question
GenSQL Query
Answer
Imputation
Guess all missing values of the "Experience" field with confidences.
Prediction
How likely is it that a developer from Seattle knows Rust?
Synthetic Data Generation
Generate 1,000 synthetic rows for women in MA.
Anomaly Detection
Who are the 3 most anomalous respondents?
Data Table
program synthesis (optional)
GenSQL Planner
Probabilistic Program

```
SELECT exp AS Experience,
  PROBABILITY OF exp UNDER
  model GIVEN * AS Pr_Exp
FROM data GENERATIVE JOIN
model GIVEN *
```

```
SELECT
  PROBABILITY OF Rust
UNDER model
GIVEN city = "Seattle"
```

```
SELECT age, salary FROM
GENERATE UNDER model
GIVEN gender = "Woman"
AND state = "MA"
LIMIT 1000
```

```
SELECT *, PROBABILITY OF
  * AS Pr_Data
UNDER model FROM data
ORDER BY ASC Pr_Data
LIMIT 3
```

Experience	Pr_Exp
5 years	0.6
3 years	0.3
1 year	0.1

value	probability
Yes	0.78
No	0.22

age	salary
26	61k
31	77k
…	…

name	…	Pr_Data
Tom	…	0.11
Lisa	…	0.15
John	…	0.27

Fig. 1. Overview of GenSQL.

1 INTRODUCTION

Building generative models of tabular data is a central focus in Bayesian data analysis [28], probabilistic machine learning [55] and in applications such as econometrics [4], healthcare [38] and systems biology [83]. Motivated by these applications, researchers have developed techniques for automatically learning rich probabilistic models of tabular data [1, 30, 36, 50, 69]. To fully exploit these models for solving complex tasks, users must be able to easily interleave operations that access both tabular data records and probabilistic models. Examples computations include (i) generating synthetic data records that satisfy user constraints; (ii) conditioning distributions specified by probabilistic models given observed data records; and (iii) using database operations to aggregate the results of combined queries against tabular and model data. However, the majority of existing probabilistic programming systems are designed for specifying generative models and estimating parameters given observations. They do not support complex database queries that combine tabular data with generative models specified by probabilistic programs.

GenSQL. This article introduces GenSQL, a novel probabilistic programming system for querying generative models of database tables. GenSQL is structured as a declarative extension to SQL which seamlessly enables queries that integrate access to the tabular data with operations against the probabilistic model. Examples include predicting new data, detecting anomalies, imputing missing values, cleaning noisy entries, and generating synthetic observations [25, 29, 46, 73]. GenSQL introduces a novel interface and soundness guarantees that decouple user-level specification of high-level queries against probabilistic models from low-level details of probabilistic programming, such as probabilistic modelling, inference algorithm design, and high-performance machine implementations. GenSQL extends SQL with several constructs:

- To complement SELECT clauses that *retrieve* existing records from a table, GenSQL includes the clause GENERATE UNDER m to *generate* synthetic records from a probabilistic model m.
- To complement WHERE clauses that *filter* data via constraints, GenSQL introduces the clause m GIVEN e to *condition* a probabilistic model m on an event (i.e., a set of constraints) e.
- To complement *joins* between tables, GenSQL introduces a new *mixed join* clause t GENERATIVE JOIN m to join each row of a data table t with a synthetic row generated from a probabilistic model m, whose generation can be conditioned in a per-row fashion on the values of t.
- To complement arithmetic expressions, GenSQL introduces PROBABILITY OF e UNDER m expressions, which compute the probability (density) of an event e under a probabilistic model m.

In this work, we assume that an existing probabilistic program synthesis tool has been used to automatically generate a probabilistic model of the user's data satisfying a certain formal interface. The user then uploads the data and model to GenSQL which automatically integrates them. The user can then issue queries for a variety of tasks, as illustrated in Fig. 1. Although we envision most users using automatically discovered models on their data, the GenSQL implementation also supports hand-implemented or partly-learned probabilistic models. For instance, a user can develop custom models for harmonization across different sources, as shown in Appendix A.3.

The core of GenSQL is formalized as a simply-typed extension of SQL (Section 3.1). This extension includes standard SQL scalar expressions and tables as well as *rowModels* (probabilistic models of tables) and *events* (a set of constructs that allow users to issue probabilistic queries that leverage Bayesian conditioning). Together, rowModels and events enable a seamless integration of standard SQL databases with probabilistic models, which include queries that interleave accesses to the database records and probabilistic models.

The GenSQL *query planner* (Section 4) lowers queries into plans that execute against a new model interface for probabilistic models of tabular data. This *Abstract Model Interface* (AMI) (Section 4.1) provides a unifying specification of probabilistic models that are compatible with GenSQL. To implement the AMI, the model must be able to: (i) generative samples from a (potentially approximate) conditional distribution; (ii) compute probability densities for specified points; (iii) compute probabilities of sets in the support of the conditional distribution.

The open source GenSQL system includes a number of implementations of the AMI, including

- a Clojure implementation [61] of Gen [20], a general purpose probabilistic programming language; see Appendix A.3 for an example.
- models produced by CrossCat [50], a probabilistic program synthesis tool;
- SPPL [73], a probabilistic programming language for exact inference.

We provide a measure-theoretic denotational semantics for the language (Section 3.2). This semantics captures the interaction between deterministic SQL operations and probabilistic operations on the probabilistic model, enabling us to prove several correctness guarantees that query results satisfy. Specifically, we prove guarantees for (i) the *exact* case, where exact inference about marginal and conditional distributions of the probabilistic model is available (Theorem 4.2); and (ii) a range of *approximate* cases, where answers to marginal and conditional queries are obtained via approximate inference algorithms (Theorem 4.3).

We benchmark GenSQL on a set of representative queries, testing the runtime performance, overhead of the query planner, and effect of our optimizations. The results show that all queries execute in milliseconds against data tables of sizes up to 10,000 rows, with a speedup in the range 1.7–6.8x against the most closely related baseline, and that the query planner's overhead as compared to hand-written code is small. We evaluate our system on two case studies to test its applicability to solving real-world problems (conditional synthetic data generation for a virtual wet lab and an anomaly detection in clinical trials), comparing against a generalized linear model (GLM) and a conditional tabular generative adversarial network (CTGAN [87]) baseline.

Contributions. This paper makes the following contributions:

(1) The **GenSQL language** (Section 3.1), an extension of SQL with probabilistic models of tabular data as first-class constructs and probabilistic constructs to allow the integration of queries on these models with queries on the data.
(2) A **unifying abstract interface for models of tabular data** (Section 4.1), which bridges the query language and probabilistic models of database tables, to which all models must conform. The query planner lowers GenSQL queries on models to queries on this interface.
(3) **Soundness theorems**, which fall into two classes:

- **Exact:** We show that if models satisfy the exact interface, all deterministic computations will be exact (Theorem 4.2). This theorem works with an exact denotational semantics (Section 4.3) that precisely characterize the behavior of exact models.
- **Approximate:** If approximate models implement consistent estimators (i.e., estimators that converge to the true value), we prove that all queries return consistent results (Theorem 4.3). This theorem works with a novel denotational semantics that combines measure-theoretic aspects with sequences of random variables.

Together, these guarantees highlight some of the tradeoffs between using an exact model (which deliver stronger guarantees but may be difficult to obtain in some use cases) and an approximate model (which deliver weaker guarantees but are more easily available).

(4) An **open-source implementation** of GenSQL in Clojure (https://github.com/OpenGen/GenSQL.query), which can be compiled into JavaScript and run natively in the browser.

(5) A **performance evaluation** of our approach (Section 5) which establishes that GenSQL is competitive with hand-coded implementations and gives improved performance over a competitive baseline. Two case studies further demonstrate the utility of GenSQL.

2 EXAMPLE

Figure 2 presents an example GenSQL query. In this example, we work with a probabilistic model (`health_model`) derived from a national database of patient information, as well as a data table (`health_data`) from a set of local hospitals. The query uses the probabilistic model to estimate the mutual information—an information-theoretic measure used in data analysis— between the `age` and `bmi` columns (from the probabilistic model) for specific valueso f patient weights (selected from the data table). The mutual information is a statistical measure of the strength of the association between these two columns, defined as a sum or integral, over the joint distribution of `age` and `bmi`, of the logarithm of the ratio of the joint density and the product of the marginal density.

The query estimates the mutual information by Monte Carlo integration, i.e., it approximates the integral by sampling. We first generate 1000 copies of each row in the `health_data` table (line 15) and then use the GenSQL *generative join* construct (line 16) to complete each row as follows. For each such row *r*:

```
1  SELECT weight, AVG(log_pxy_div_px_py) AS mutual_information
2  FROM (
3    SELECT weight, LOG(pxy) - (LOG(px) + LOG(py)) AS log_pxy_div_px_py
4    FROM (
5      SELECT weight,
6        PROBABILITY OF h_model.age = table.age AND h_model.bmi = table.bmi
7          UNDER h_model GIVEN h_model.weight = table.weight AS pxy,
8        PROBABILITY OF h_model.age = table.age
9          UNDER h_model GIVEN h_model.weight = table.weight AS px,
10       PROBABILITY OF h_model.bmi = table.bmi
11         UNDER h_model GIVEN h_model.weight = table.weight AS py
12     FROM (
13       SELECT table.weight, table.age, table.bmi
14       FROM (
15         health_data DUPLICATE 1000 TIMES
16         GENERATIVE JOIN h_model
17         GIVEN h_model.weight = health_data.weight) AS table)))
18 GROUP BY weight
```

Fig. 2. Estimating the conditional mutual information between age and bmi given patient weights.

(1) a row r' is sampled from a version of the model conditioned on the `weight` value of row r;
(2) the rows r and r' are concatenated.

The resulting intermediate table is called `table` (line 17). Each synthetic row r' is used as a sample for the Monte Carlo integration of the conditional mutual information for the corresponding `weight` value. From this intermediate table, we select the `weight`, `age`, and `bmi` columns (line 13). Note that the `weight` column comes from the patient data while the `age` and `bmi` columns come from the rows sampled from the probabilistic model.

For each `weight` in the patient data, we compute the Monte Carlo approximation of the mutual information between `age` and `bmi` for that weight as

$$\frac{1}{1000k} \sum_{i=1}^{1000k} \log \frac{p(\texttt{age}_i,\texttt{bmi}_i)}{p(\texttt{age}_i)p(\texttt{bmi}_i)},$$

where k is the number of patients with that specific weight, and $(\texttt{age}_i, \texttt{bmi}_i)$ is a sample from the model for that weight. To do so, lines 6–11 compute the probability densities $p(\texttt{age}_i, \texttt{bmi}_i)$ (lines 6–7), $p(\texttt{age}_i)$ (lines 8-9), and $p(\texttt{bmi}_i)$ (lines 10–11). For instance, the `GIVEN` clause conditions the model on the `weight` column of the model being equal to the `weight` column of `table` (line 11). Line 10 then computes the probability density that the `bmi` column of the conditioned model is equal to the corresponding column in `table`. GenSQL computes these probability densities by invoking the **logpdf** function in the probabilistic model interface (Section 4.1).

A traditional SQL select statement (line 5) propagates the patient weights and corresponding probabilities `pxy`, `px`, and `py` to generate a table with four columns: the `weight` and the corresponding probability densities for that weight. Line 3 computes $\log p(\texttt{age}_i, \texttt{bmi}_i) - \log p(\texttt{age}_i)p(\texttt{bmi}_i)$ for each of the rows, naming this ratio `log_pxy_div_px_py`. Note that there are $1000k$ `log_pxy_div_px_py` values for each weight in the local patient data, where k is the number of patients with that specific weight. Finally, line 1 computes the mutual information estimate between `age` and `bmi`, for each weight, as the average of the `log_pxy_div_px_py` values for that weight. This example illustrates the expressivity of GenSQL, but we note that our implementation has a primitive which directly estimates conditional mutual information without the need to materialize intermediate tables.

3 SYNTAX AND SEMANTICS

3.1 Language

The core calculus extending SQL for querying from probabilistic models of tabular data is given in Fig. 3, and the type-system is given in Fig. 4.[1] As SQL is a subset of GenSQL, this calculus also includes a simply-typed formalization of SQL where terms are given in a pair of context: a local and a global one. We found this formalization interesting in its own right, as we could not find an equivalent formalization in the programming languages literature.

There are several noteworthy differences with variables and contexts from traditional simply-typed languages based on the lambda-calculus, which are explained below. We note early that we distinguish two types of conditioning through constructs called events and events-0. Events-0 come from a technical difficulty well-known in the PPL and measure theory literature [77, 85] when conditioning on a continuous variable taking a specific value. This creates a *possible event of probability 0*, and requires special treatment.

Names and Identifiers. We assume a countable set C of names $\textsc{col} \in C$ for the columns of tables and rowModels, as well as a countable set I of identifiers $\textsc{id} \in I$ for naming tables and rowModels.

[1]The core SQL-part of the language is minimal for expository purposes. Appendix E presents the formalization for a richer language, including the operations **GROUP BY** and **DUPLICATE** used in Fig. 2.

Description	SQL	Probabilistic Extension
Base/Event Type	$\sigma \quad ::= \quad \sigma_c \mid \sigma_d$	$\mathcal{E} \quad ::= \quad C^1\{\text{COL}_1 : \sigma_1, \ldots, \text{COL}_n : \sigma_n\}$ $\mid C^0\{\text{COL}_1 : \sigma_1, \ldots, \text{COL}_n : \sigma_n\}$
Table/RowModel Type	$\mathcal{T} \quad ::= \quad T[\text{ID}]\{\text{COL}_1 : \sigma_1, \ldots, \text{COL}_n : \sigma_n\}$	$\mathcal{M} \quad ::= \quad M[\text{ID}]\{\text{COL}_1 : \sigma_1, \ldots, \text{COL}_n : \sigma_n\}$
Table Expression	$t \quad ::- \quad \text{ID} \mid \textbf{RENAME } t \textbf{ AS } \text{ID}$ $\mid t_1 \textbf{ JOIN } t_2 \mid t \textbf{ WHERE } e$ $\mid \textbf{SELECT } \bar{e} \textbf{ AS } \overline{\text{COL}} \textbf{ FROM } t$	$t \quad ::= \quad \ldots$ $\mid \textbf{GENERATE UNDER } m \textbf{ LIMIT } e$ $\mid t \textbf{ GENERATIVE JOIN } m$
RowModel Expression		$m \quad ::= \quad \text{ID} \mid m \textbf{ GIVEN } c^i \mid \textbf{RENAME } m \textbf{ AS } \text{ID}$
Scalar Expression	$e \quad ::= \quad \text{ID.COL} \mid op(e_1, \ldots, e_n)$	$e \quad ::= \quad \ldots \mid \textbf{PROBABILITY OF } c^i \textbf{ UNDER } m$
Event Expressions Event-0 Expressions		$c^1 \quad ::= \quad c^1_1 \wedge c^1_2 \mid c^1_1 \vee c^1_2 \mid \text{ID.COL } op\ e$ $c^0 \quad ::= \quad c^0_1 \wedge c^0_2 \mid \text{ID.COL} = e$
Primitive Domains:	$op \in \textbf{Op}$, $\text{ID} \in \mathcal{I}$, $\text{COL} \in C$, $\sigma_c \in \{\textbf{Real}, \textbf{PosReal}, \textbf{Ranged}(a, b), \ldots\}$, $\sigma_d \in \{\textbf{Int}, \textbf{Str}, \textbf{Nat}, \textbf{Bool}, \ldots\}$	
Syntactic Sugar:	$\bar{e} \textbf{ AS } \overline{\text{COL}} \equiv e_1 \textbf{ AS } \text{COL}_1, \ldots, e_n \textbf{ AS } \text{COL}_n.$	

Fig. 3. Syntax of GenSQL.

Base types. Cells of tables can have a base type σ, which is either a continuous type σ_c or a discrete type σ_d. Continuous types are **Real** for reals, **PosReal** for non-negative reals, or **Ranged**(a, b) for reals in the range $[a, b]$. Discrete types are **Nat** for natural numbers, **Int** for integers, **Str** for strings, **Cat**$(\text{N}_1, \ldots, \text{N}_k)$ for a categorical type over k attributes, and **Bool** for Boolean.

Table and rowModel types. We denote these types by $D[?\text{ID}]\{\text{COL}_1 : \sigma_1, \ldots, \text{COL}_n : \sigma_n\}$. D is either T for tables or M for rowModels. ?ID is an optional identifier, allowing access to columns of a table or rowModel in a query. For instance, in SELECT ID.weight, the identifier ID refers to a table and weight to a column of that table. The identifier can be be optional, e.g. there is no default identifier for a table created after a join. The notation $\{\text{COL}_1 : \sigma_1, \ldots, \text{COL}_n : \sigma_n\}$ indicate that the table has columns COL_i of type σ_i. Therefore, we can think of each row of a table as an element of a record type $\{\text{COL}_1 : \sigma_1, \ldots, \text{COL}_n : \sigma_n\}$, a bag of rows as a table, and a rowModel as a row generator.

Scalar Expressions. **Op** is a set of primitive operations on base types including standard operations such as $+, *, <, >, =$ on integers and reals, $\wedge, \vee$ on Booleans, as well as constants for every value of a base type. For any $op \in \textbf{Op}$, we write $op : \sigma_1, \ldots, \sigma_n \longrightarrow \sigma$ if op has arity n, takes arguments of base types $\sigma_1, \ldots, \sigma_n$ and returns a value of type σ. In particular, operations with no arguments are constants of the appropriate type such as **true** and **false** at the boolean type. All base types have an additional constant **Null** representing a missing value. This constant is preserved by primitive operations (e.g. **Null** $+ 3 \mapsto$ **Null**, **Null** $* 4.1 \mapsto$ **Null**). By convention, WHERE **Null** clauses act as WHERE **false**.

Table expressions. Apart from typical SQL operations, we have two ways to generate synthetic data. GENERATE UNDER returns a synthetic table with a given number of rows specified by the LIMIT clause, where each row is generated by sampling from a given rowModel. GENERATIVE JOIN takes a rowModel and a table, and returns a synthetic table with the same number of rows, where each row is generated by concatenating the current row of the table with a sample from the rowModel. The model generating the samples can be conditioned on the current row of the table. RENAME renames a table or rowModel with a new identifier, therefore changing the identifier in its type and the way to access their column in a select of event clause.

Event and event-0 expressions. Events are Boolean expressions on tables and rowModels, which include equality on discrete values but not on continuous values, which is reserved for events-0. The only probability 0 events are impossible under a given model, e.g. $x > 6 \wedge x < 3$, and those do not require a separate treatment. Events and events-0 are used in the PROBABILITY OF and GIVEN constructs.

(a) **Type System for Table Expressions**

$$\frac{}{\Gamma, T[\textsc{id}]\{\textsc{cols}\};\Delta \vdash \textsc{id} : T[\textsc{id}]\{\textsc{cols}\}}$$

$$\frac{\Gamma;\Delta \vdash t : T[?\textsc{id}]\{\textsc{cols}\} \quad \textsc{id}' \text{ fresh}}{\Gamma;\Delta \vdash \ \textbf{RENAME}\ t\ \textbf{AS}\ \textsc{id}' : T[\textsc{id}']\{\textsc{cols}\}}$$

$$\frac{\begin{array}{c}\Gamma;\Delta \vdash t_1 : T[?\textsc{id}_1]\{\textsc{cols}\} \\ \Gamma;\Delta \vdash t_2 : T[?\textsc{id}_2]\{\textsc{cols}'\} \quad \textsc{cols} \cap \textsc{cols}' = \emptyset\end{array}}{\Gamma;\Delta \vdash t_1\ \textbf{JOIN}\ t_2 : T[\,]\{\textsc{cols}, \textsc{cols}'\}}$$

$$\frac{\Gamma;\Delta \vdash t : T[\textsc{id}]\{\textsc{cols}\} \quad \Gamma;\Delta, T[\textsc{id}]\{\textsc{cols}\} \vdash e : \textbf{Bool}}{\Gamma;\Delta \vdash t\ \textbf{WHERE}\ e : T[\textsc{id}]\{\textsc{cols}\}}$$

$$\frac{\Gamma;\Delta \vdash m : M[\textsc{id}]\{\textsc{cols}\} \quad \Gamma;\Delta \vdash e : \textbf{Nat}}{\begin{array}{c}\Gamma;\Delta \vdash \ \textbf{GENERATE UNDER}\ m \\ \textbf{LIMIT}\ e : T[\,]\{\textsc{cols}\}\end{array}}$$

$$\frac{\begin{array}{c}\Gamma;\Delta \vdash t : T[\textsc{id}]\{\textsc{cols}\} \quad \textsc{cols} \cap \textsc{cols}' = \emptyset \\ \Gamma;\Delta, T[\textsc{id}]\{\textsc{cols}\} \vdash m : M[\textsc{id}']\{\textsc{cols}'\}\end{array}}{\Gamma;\Delta \vdash t\ \textbf{GENERATIVE JOIN}\ m : T[\,]\{\textsc{cols}, \textsc{cols}'\}}$$

$$\frac{\begin{array}{c}\Gamma;\Delta \vdash t : T[\textsc{id}]\{\textsc{cols}\} \\ \Gamma;\Delta, T[\textsc{id}]\{\textsc{cols}\} \vdash e_i : \sigma_i \text{ for } 1 \leq i \leq n \\ \bar{e}\ \textbf{AS}\ \overline{\textsc{col}} := e_1\ \textbf{AS}\ \textsc{col}_1, \dots, e_n\ \textbf{AS}\ \textsc{col}_n\end{array}}{\begin{array}{c}\Gamma;\Delta \vdash \ \textbf{SELECT}\ \bar{e}\ \textbf{AS}\ \overline{\textsc{col}} \\ \textbf{FROM}\ t : T[\,]\{\textsc{col}_1 : \sigma_1, \dots, \textsc{col}_n : \sigma_n\}\end{array}}$$

(b) **Type System for Event Expressions**

$$\frac{\begin{array}{c}\Gamma;\Delta \vdash e : \sigma_i \quad i \in \{1, \dots, n\} \\ op \in \{<, >, =\} \quad \forall \sigma_c.(\sigma_i, op) \neq (\sigma_c, =) \\ \textsc{cols} = \textsc{col}_1 : \sigma_1, \dots, \textsc{col}_i : \sigma_i, \dots, \textsc{col}_n : \sigma_n\end{array}}{\Gamma;\Delta, M[\textsc{id}]\{\textsc{cols}\} \vdash \textsc{id}.\textsc{col}_i\ op\ e : C^1\{\textsc{cols}\}}$$

$$\frac{\Gamma;\Delta \vdash c^1_1 : C^1\{\textsc{cols}\} \quad \Gamma;\Delta \vdash c^1_2 : C^1\{\textsc{cols}\}}{\Gamma;\Delta \vdash c^1_1 \wedge c^1_2 : C^1\{\textsc{cols}\}}$$

$$\frac{\Gamma;\Delta \vdash c^1_1 : C^1\{\textsc{cols}\} \quad \Gamma;\Delta \vdash c^1_2 : C^1\{\textsc{cols}\}}{\Gamma;\Delta \vdash c^1_1 \vee c^1_2 : C^1\{\textsc{cols}\}}$$

(c) **Type System for rowModel Expressions**

$$\frac{}{\Gamma, M[\textsc{id}]\{\textsc{cols}\};\Delta \vdash \textsc{id} : M[\textsc{id}]\{\textsc{cols}\}}$$

$$\frac{\begin{array}{c}\Gamma;\Delta \vdash m : M[\textsc{id}]\{\textsc{cols}\} \\ \Gamma;\Delta, M[\textsc{id}]\{\textsc{cols}\} \vdash c^1 : C^1\{\textsc{cols}\}\end{array}}{\Gamma;\Delta \vdash m\ \textbf{GIVEN}\ c^1 : M[\textsc{id}]\{\textsc{cols}\}}$$

$$\frac{\Gamma;\Delta, M[\textsc{id}]\{\textsc{cols}\} \vdash c^0 : C^0\{\textsc{cols}'\}}{\Gamma;\Delta \vdash \textsc{id}\ \textbf{GIVEN}\ c^0 : M[\textsc{id}]\{\textsc{cols}\}}$$

$$\frac{\Gamma;\Delta \vdash m : M[\textsc{id}]\{\textsc{cols}\} \quad \textsc{id}' \text{ fresh}}{\Gamma;\Delta \vdash \ \textbf{RENAME}\ m\ \textbf{AS}\ \textsc{id}' : M[\textsc{id}']\{\textsc{cols}\}}$$

(d) **Type System for Scalar Expressions**

$$\frac{i \in \{1, \dots, n\}}{\Gamma;\Delta, T[\textsc{id}]\{\textsc{col}_1 : \sigma_1, \dots, \textsc{col}_n : \sigma_n\} \vdash \textsc{id}.\textsc{col}_i : \sigma_i}$$

$$\frac{\begin{array}{c}\Gamma;\Delta \vdash m : M[\textsc{id}]\{\textsc{cols}\} \\ \Gamma;\Delta, M[\textsc{id}]\{\textsc{cols}\} \vdash c^1 : C^1\{\textsc{cols}\}\end{array}}{\Gamma;\Delta \vdash \ \textbf{PROBABILITY OF}\ c^1\ \textbf{UNDER}\ m : \textbf{Ranged}(0, 1)}$$

$$\frac{\begin{array}{c}\Gamma;\Delta \vdash m : M[\textsc{id}]\{\textsc{cols}\} \quad \textbf{vars}(c^0) \cap \textbf{condvars}(m) = \emptyset \\ \Gamma;\Delta, M[\textsc{id}]\{\textsc{cols}\} \vdash c^0 : C^0\{\textsc{cols}'\}\end{array}}{\Gamma;\Delta \vdash \ \textbf{PROBABILITY OF}\ c^0\ \textbf{UNDER}\ m : \textbf{PosReal}}$$

$$\frac{\Gamma;\Delta \vdash e_i : \sigma_i \text{ for } 1 \leq i \leq n \quad op : \sigma_1, \dots, \sigma_n \to \sigma}{\Gamma;\Delta \vdash op(e_1, \dots, e_n) : \sigma}$$

(e) **Type System for Event-0 Expressions**

$$\frac{\begin{array}{c}\Gamma;\Delta \vdash e : \sigma \quad i \in \{1, \dots, n\} \\ \textsc{cols} = \dots, \textsc{col}_i : \sigma, \dots\end{array}}{\Gamma;\Delta, M[\textsc{id}]\{\textsc{cols}\} \vdash \textsc{id}.\textsc{col}_i \ = \ e : C^0\{\textsc{col}_i : \sigma\}}$$

$$\frac{\begin{array}{c}\Gamma;\Delta \vdash c^0_1 : C^0\{\textsc{cols}\} \quad \Gamma;\Delta \vdash c^0_2 : C^0\{\textsc{cols}'\} \\ \textsc{cols} \cap \textsc{cols}' = \emptyset\end{array}}{\Gamma;\Delta \vdash c^0_1 \wedge c^0_2 : C^0\{\textsc{cols}, \textsc{cols}'\}}$$

Fig. 4. Type system of GenSQL.

PROBABILITY OF takes an event (or event-0) expression and a rowModel to query and returns the probability (or probability density) of the event under the model.[2]

RowModel expressions. **GIVEN** takes a rowModel and an event (or event-0) expression, and returns a new rowModel, the conditional distribution of the original rowModel on the event. The event expression can be given by a list of inequalities on *arbitrary variables* and equalities on discrete variables, in which case **GIVEN** acts as a set of constraints on the possible returned values of the model. Otherwise, the event expression can be a set of equalities on possibly continuous values and is understood as conditioning the model on the given values.

Contexts. Expressions are typed in a pair of contexts $\Gamma; \Delta$ containing table and rowModel types. As these types include identifiers, there is no need for the more classical notation $x : \tau$ pairing a variable with its type. Γ is a *set* of types, while Δ is an *ordered list* of types. In the premise of a

[2]It may be confusing for people familiar with probabilistic programming languages (PPLs) to use **PROBABILITY OF** for both a probability mass and a probability density. Our implementation has two versions of the syntax: a strict one and a permissive one. The strict syntax distinguishes between the two, and in particular on events-0 one the primitive is **PROBABILITY DENSITY OF**. The permissive syntax allows to use **PROBABILITY OF** for both, and the system will automatically choose the right version based on the type of the event.

typing rule such as **PROBABILITY OF** , only the last element of Δ will be accessible to an expression. We denote the empty context by $[]$. Intuitively, Γ contains the ambient tables in the database schema and any loaded models, and within Γ, all identifiers $\textsc{id}$ are assumed distinct. Δ is the *value environment*, and contains only tables that are "in scope" for a particular expression. Scalar, event, and event-0 expressions all depend on the value environment. If an expression has a table in scope, it will be iterated over the rows of that table and can only access the current row. If a **PROBABILITY OF** expression has a rowModel in scope, it will query the model for the probability of an event under that model. If it's a **GIVEN** expression, it will condition that model on an event.

Typing rules. Judgments are of the form $\Gamma; \Delta \vdash \mathtt{e} : \mathtt{t}$ where $\mathtt{e}$ is an expression (t, e, m, c^1, c^0 in Figure 3); $\mathtt{t}$ is a type (σ, $\mathcal{T}$, $\mathcal{E}$, $\mathcal{M}$ in Figure 3), and $\Gamma; \Delta$ is a context. Given some loaded tables and rowModels forming environment Γ, the objects of interest are "closed expressions" of table type, i.e., expressions of the form $\Gamma; [] \vdash t : T[?\textsc{id}]\{\textsc{cols}\}$. "Closed" here refers to Δ being empty, not Γ. Notable rules include those that need the same identifier twice, such as the **PROBABILITY OF** or the **WHERE** rule. For instance, in the t **WHERE** e rule, where t has identifier $\textsc{id}$, a valid SQL e would be $\textsc{id}.\textsc{col} = 3$ where $\textsc{col}$ is a column of t. This reflects the fact that the expression e should have access to the identifier $\textsc{id}$ in its local environment, and that the column $\textsc{col}$ of t will be iterated over by the expression e.

Notations used in the type system. $?\textsc{id}$ indicates an optional identifier and $\textsc{id}$. "$\textsc{id}'$ fresh" means that $\textsc{id}'$ is not in the contexts Γ, Δ or in the type of a subterm of the expression. We will often abbreviate $\{\textsc{col}_1 : \sigma_1, \ldots, \textsc{col}_n : \sigma_n\}$ as $\{\textsc{cols}\}$. We write $\textsc{cols} \cap \textsc{cols}' = \emptyset$ when the set of column names in $\textsc{cols}$ and in $\textsc{cols}'$ should be disjoint. In the first typing rule for events, we write $\forall \sigma_c.(\sigma_i, op) \neq (\sigma_c, =)$ to mean that op cannot be an equality on a continuous type. We recursively define the following two macros:

$$\mathbf{vars}(\textsc{id}.\textsc{col}\ op\ t) = \{\textsc{col}\} \quad \mathbf{vars}(c \wedge c') = \mathbf{vars}(c) \cup \mathbf{vars}(c') \quad \mathbf{vars}(c \vee c') = \mathbf{vars}(c) \cup \mathbf{vars}(c')$$

$$\mathbf{condvars}(m\ \textbf{GIVEN}\ c^0) = \mathbf{vars}(c^0) \quad \mathbf{condvars}(m\ \textbf{GIVEN}\ c^1) = \mathbf{condvars}(m) \quad \mathbf{condvars}(\textsc{id}) = \emptyset$$

Restrictions imposed by the type system. If the same identifier $\textsc{id}$ appears twice in the premise of a typing rule, the two identifiers must equal, and two different identifiers $\textsc{id}$ and $\textsc{id}'$ must be distinct. The **JOIN** and **GENERATIVE JOIN** operations require that the columns of the two tables have disjoint names. As explained above, events are disallowed to be equalities on continuous types. A model can only be conditioned once on an event-0, which is enforced by the restriction $\textsc{id}$ **GIVEN** c^0. Events-0 follow a linear typing system to avoid contradictory statements such as $\textsc{id}.\textsc{col} = 1.0 \wedge \textsc{id}.\textsc{col} = 2.0$. Events-0 in a **PROBABILITY OF** query on a conditioned model cannot refer to the conditioned columns of the model, which is enforced by the restriction $\mathbf{vars}(c^0) \cap \mathbf{condvars}(m) = \emptyset$.[3]

Syntactic sugar. Our implementation includes various syntactic sugars that are not present in the formalization but which are used in several figures. Given $t{:}T[\textsc{id}]\{\textsc{col}_1{:}\sigma_1, \ldots, \textsc{col}_n{:}\sigma_n\}$, $m{:}M[\textsc{id}']\{\textsc{col}'_1{:}\sigma_1, \ldots, \textsc{col}'_n{:}\sigma_n\}$, we have the following equivalences:

- **SELECT** $*$ **FROM** $t \rightsquigarrow$ **SELECT** $\textsc{id}.\textsc{col}_1, \ldots, \textsc{id}.\textsc{col}_n$ **FROM** t
- **PROBABILITY OF** $* \rightsquigarrow$ **PROBABILITY OF** e for any query of the form **SELECT PROBABILITY OF** $*$ **UNDER** m **GIVEN** c **FROM** t, where $e \coloneqq \textsc{id}'.\textsc{col}'_1 = \textsc{id}.\textsc{col}_1 \wedge \ldots \wedge \textsc{id}'.\textsc{col}'_n = \textsc{id}.\textsc{col}_n$.
- m **GIVEN** $* \rightsquigarrow m$ **GIVEN** e within a **SELECT FROM** t query. The event $e \coloneqq \textsc{id}'.\textsc{col}'_{i_1} = \textsc{id}.\textsc{col}_{i_1} \wedge \ldots \wedge \textsc{id}'.\textsc{col}'_{i_k} = \textsc{id}.\textsc{col}_{i_k}$, where the $\textsc{col}_{i_j}$ are columns t that do not appear in the **SELECT** clause.
- $*$ **EXCEPT** $\textsc{id}.\textsc{col}$ removes the column $\textsc{col}$ from list of columns that $*$ selects.

[3]Our implementation is less restricted. It allows join variants such as SQL's left join where the tables do not have disjoint columns. It also allows multiple conditionings on the same model, which are then normalized to the form above. See Appendix D.1 for details about the normalization.

3.2 Semantics

We define denotational semantics using measure theory, shown in Fig. 5. Even though the SQL subset of GenSQL is not probabilistic, our probabilistic semantics ensures compositional reasoning about the semantics of SQL queries combined with probabilistic GenSQL expressions, such as synthetic tables generated by rowModels. Per usual, the semantics of expressions is defined compositionally on typing judgement derivations, and $[\![\mathtt{e}]\!]$ is a shorthand for $[\![\Gamma; \Delta \vdash \mathtt{e} : \mathtt{t}]\!]$.

Base types (Fig. 5c). We assign to each type σ a measure space $[\![\sigma]\!] := (X_\sigma, \Sigma_{X_\sigma}, \nu_\sigma)$ consisting of a set X_σ, a sigma-algebra Σ_{X_σ}, and reference measure ν_σ. $\mathbb{Z}$ denotes the set of integers, $\mathbb{N}$ natural numbers, and $\mathbb{B}$ Booleans, which are equipped with the discrete sigma-algebra. We equip the reals $\mathbb{R}$ with the Borel sigma-algebra. We interpret **Null** by adding a fresh element $\{\star\}$ to the standard interpretation of each base type, equipped with the discrete sigma-algebra. The semantics of a base type σ is then given by the smallest sigma-algebra making $\{\star\}$ measurable, as well as ensuring that every previously measurable set remains measurable. (This construction is also called the "direct-sum sigma-algebra" [24, 214L]).

The base measure on discrete types σ_d such as **Int**, **Nat**, **Bool**, **Str** is the counting measure. On continuous types σ_c such as **Real**, the base measure is the Lebesgue measure λ. These are extended to base measures ν_σ on $[\![\sigma]\!]$ by using the dirac measure $\delta_{\{\star\}}$ on $\{\star\}$, e.g. the base measure on $[\![\mathbb{R}]\!]$ is $\lambda_\mathbb{R} + \delta_{\{\star\}}$. We write $\mu \otimes \nu$ for the product of measures. We extend the reference measure to the product space $\prod_{1\leq i\leq n}[\![\sigma_i]\!]$ by taking the product of the reference measures $\nu := \bigotimes_{1\leq i\leq n} \nu_{\sigma_i}$.

Table types (Fig. 5a). Our semantics has two modes of interpreting table types, a "tuple mode" $\mathrm{Tup}[\![-]\!]$, and a "table mode" $\mathrm{Tab}[\![-]\!]$. $\mathrm{Tab}[\![-]\!]$ interprets tables as measures on bags of tuples, while $\mathrm{Tup}[\![-]\!]$ interprets a table as a tuple, representing the current row of the table being processed by a scalar, event or event-0 expression. More precisely, we denote by $\mathcal{P}(X)$ the measurable space of probability measures on the standard Borel space X [32]. The table semantics interprets table types as measures on bags of tuples $\mathrm{Tab}[\![T[?\textsc{id}]\{\textsc{cols}\}]\!] = \mathbf{Bag}(\mathrm{Tup}[\![T[?\textsc{id}]\{\textsc{cols}\}]\!])$, where $\mathbf{Bag}(X) = \{f : X \to \mathbb{N} \mid f(x) = 0 \text{ except for finitely many } x\}$. $\mathbf{Bag}(X)$ is equipped with the least sigma-algebra containing the generating sets $\{b \in \mathbf{Bag}(X) \mid b \text{ contains exactly } k \text{ elements in } A\}$ for measurable sets A of X [21].

Contexts (Fig. 5b). We interpret the global context Γ with the table semantics $\mathrm{Tab}[\![-]\!]$ and the local context Δ with the tuple semantics $\mathrm{Tup}[\![-]\!]$. We write γ for an element of $\mathrm{Tab}[\![\Gamma]\!]$, and see it as a finite map from identifiers to values. Likewise, we write δ for an element of Δ. We write $\delta[\textsc{id} \mapsto v]$ for the extended finite map mapping $\textsc{id}$ to v.

Scalar expressions (Fig. 5d). We then interpret scalar expressions $\Gamma; \Delta \vdash e : \sigma$ as measurable functions $\mathrm{Tab}[\![\Gamma]\!] \times \mathrm{Tup}[\![\Delta]\!] \to [\![\sigma]\!]$. We lift operations op to interpret **Null**, and write op_s for the extended version of op which sends $\star$ to $\star$.

Event expressions (Fig. 5g). An event expression $[\![c^1 : C^1\{\textsc{cols}\}]\!](\gamma, \delta)$ is interpreted as a measurable subset S of $[\![\textsc{cols}]\!]$ (disjoint union of hyper-rectangles [73]). Depending on the expression, this set S is used in different ways. We interpret the probability clause PROBABILITY OF c^1 UNDER m as $\int_{[\![\textsc{cols}]\!]} \mathbb{1}_S d\mu$, where μ is the measure denoting the model m, i.e. S is used in an indicator function $\mathbb{1}_S$. When used in a GIVEN clause, we constrain the model to the event S, which is then renormalized. If the event has probability 0, we instead return a row of **Null**. A similar situation to WHERE **Null** arises for GIVEN , e.g. in GIVEN $\textsc{id.col}$ op **Null**. Following the principle of least surprise, **Null** acts by convention as a unit for conditioning, i.e. $\textsc{id}$ GIVEN $\textsc{id.col}$ op **Null** behaves the same as $\textsc{id}$. To ensure this we interpret boolean expressions op differently in the semantics of events, and write op_l for the extended version of op which sends $\star$ to **true**. The denotation of $\textsc{id.col}$ op **Null** will therefore be the entire space, and conditioning a model on this event will not change its denotation.

(a) **Semantics of Table and rowModel Types**

$$\mathsf{Tup}\llbracket T[?\textsc{id}]\{\textsc{col}_1 : \sigma_1, \ldots, \textsc{col}_n : \sigma_n\}\rrbracket = \textstyle\prod_{1\le i\le n}\llbracket\sigma_i\rrbracket$$

$$\llbracket M[?\textsc{id}]\{\textsc{col}_1 : \sigma_1, \ldots, \textsc{col}_n : \sigma_n\}\rrbracket = \mathcal{P}_{\text{dens}}\Big(\textstyle\prod_{1\le i\le n}\llbracket\sigma_i\rrbracket\Big)$$

$$\mathsf{Tab}\llbracket T[?\textsc{id}]\{\textsc{col}_1 : \sigma_1, \ldots, \textsc{col}_n : \sigma_n\}\rrbracket = \mathcal{P}\mathbf{Bag}\Big(\textstyle\prod_{1\le i\le n}\llbracket\sigma_i\rrbracket\Big)$$

$$\mathsf{Tab}\llbracket M[?\textsc{id}]\{\textsc{col}_1 : \sigma_1, \ldots, \textsc{col}_n : \sigma_n\}\rrbracket = \mathcal{P}_{\text{adm}}\Big(\textstyle\prod_{1\le i\le n}\llbracket\sigma_i\rrbracket\Big)$$

(b) **Semantics of Contexts**

$$\mathsf{Tup}\llbracket\Delta := \mathcal{T}, \Delta'\rrbracket = \mathsf{Tup}\llbracket\mathcal{T}\rrbracket \times \mathsf{Tup}\llbracket\Delta'\rrbracket$$

$$\mathsf{Tup}\llbracket\Delta := \mathcal{M}, \Delta'\rrbracket = \llbracket\mathcal{M}\rrbracket \times \mathsf{Tup}\llbracket\Delta'\rrbracket$$

$$\mathsf{Tab}\llbracket\Gamma := \mathcal{T}, \Gamma'\rrbracket = \mathsf{Tab}\llbracket\mathcal{T}\rrbracket \times \mathsf{Tab}\llbracket\Gamma'\rrbracket$$

$$\mathsf{Tab}\llbracket\Gamma := \mathcal{M}, \Gamma'\rrbracket = \mathsf{Tab}\llbracket\mathcal{M}\rrbracket \times \mathsf{Tab}\llbracket\Gamma'\rrbracket$$

(c) **Semantics of Base Types**

$$\llbracket\mathbf{Bool}\rrbracket := (\mathbb{B}\cup\{\star\}, \mathcal{P}(\mathbb{B}\cup\{\star\}), \nu_{\mathbb{B}})$$

$$\llbracket\mathbf{Int}\rrbracket := (\mathbb{Z}\cup\{\star\}, \mathcal{P}(\mathbb{Z}\cup\{\star\}), \nu_{\mathbb{Z}})$$

$$\llbracket\mathbf{Str}\rrbracket := (\mathbf{Str}\cup\{\star\}, \mathcal{P}(\mathbf{Str}\cup\{\star\}), \nu_{\mathbf{Str}})$$

$$\llbracket\mathbf{Real}\rrbracket := (\mathbb{R}\cup\{\star\}, \mathcal{B}(\mathbb{R}\cup\{\star\}), \nu_{\mathbb{R}})$$

$$\llbracket\mathbf{PosReal}\rrbracket := (\mathbb{R}^+\cup\{\star\}, \mathcal{B}(\mathbb{R}^+\cup\{\star\}), \nu_{\mathbb{R}^+})$$

(d) **Semantics of Scalar Expressions**

$$\llbracket\textsc{id}.\textsc{col}_i\rrbracket(\gamma,\delta) = \pi_i(\delta(\textsc{id})) \quad \text{where } T[?\textsc{id}]\{\textsc{cols}\} \in \Delta$$

$$\llbracket op(e_1,\ldots,e_n)\rrbracket(\gamma,\delta) = op_s\,(\llbracket e_1\rrbracket(\gamma,\delta),\ldots,\llbracket e_n\rrbracket(\gamma,\delta))$$

$$\llbracket\ \mathbf{PROBABILITY\ OF}\ c^1\ \mathbf{UNDER}\ m\rrbracket(\gamma,\delta) = \llbracket m\rrbracket(\gamma,\delta).\text{meas}\big(\llbracket c^1\rrbracket(\gamma,\delta)\big)$$

$$\llbracket\ \mathbf{PROBABILITY\ OF}\ c^0\ \mathbf{UNDER}\ m\rrbracket(\gamma,\delta) = \mathbf{let}\ (\pi, v) = \llbracket c^0\rrbracket(\gamma,\delta[\textsc{id}\mapsto\llbracket m\rrbracket(\gamma,\delta)])\,\mathbf{in}\ \llbracket m\rrbracket(\gamma,\delta).\text{pdf}(v)$$

(e) **Semantics of Table Expressions**

$$\llbracket\textsc{id} : T[?\textsc{id}]\{\textsc{cols}\}\rrbracket(\gamma,\delta) = \gamma(\textsc{id}) \qquad \llbracket\ \mathbf{RENAME}\ t\ \mathbf{AS}\ \textsc{id}'\rrbracket(\gamma,\delta) = \llbracket t\rrbracket(\gamma,\delta)$$

$$\llbracket t_1\ \mathbf{JOIN}\ t_2\rrbracket(\gamma,\delta) = (\lambda x, y.\mathbf{map2}\ (\lambda r_1, r_2.(r_1, r_2))\ x\ y)_*(\llbracket t_1\rrbracket(\gamma,\delta)\otimes\llbracket t_2\rrbracket(\gamma,\delta))$$

$$\llbracket t : T[?\textsc{id}]\{\textsc{cols}\}\ \mathbf{WHERE}\ e\rrbracket(\gamma,\delta) = \Big(\lambda x.\mathbf{filter}\ (\lambda r.\llbracket e\rrbracket(\gamma,\delta[\textsc{id}\mapsto r]))\ x\Big)_*\llbracket t\rrbracket(\gamma,\delta)$$

$$\llbracket\ \mathbf{GENERATE\ UNDER}\ m\ \mathbf{LIMIT}\ e\rrbracket(\gamma,\delta) = \mathbf{let}\ n = \llbracket e\rrbracket(\gamma,\delta)\ \mathbf{in}\ \Big(\lambda(x_1,\ldots,x_n).\textstyle\bigcup_{1\le i\le n}\{x_i\}\Big)_*\textstyle\bigotimes_{1\le i\le n}\llbracket m\rrbracket(\gamma,\delta).\text{meas}$$

$$\llbracket\ \mathbf{SELECT}\ e_1\ \mathbf{AS}\ \textsc{col}_1,\ldots,e_n\ \mathbf{AS}\ \textsc{col}_n\ \mathbf{FROM}\ t : T[?\textsc{id}]\{\textsc{cols}\}\rrbracket(\gamma,\delta) = \Big(\lambda x.\mathbf{map}\ (\lambda r.(\llbracket e_1\rrbracket(\gamma,\delta[\textsc{id}\mapsto r]),\ldots,\llbracket e_n\rrbracket(\gamma,\delta[\textsc{id}\mapsto r])))\ x\Big)_*\llbracket t\rrbracket(\gamma,\delta)$$

$$\llbracket(t : T[?\textsc{id}]\{\textsc{cols}\})\ \mathbf{GENERATIVE\ JOIN}\ m\rrbracket(\gamma,\delta) = \llbracket t\rrbracket(\gamma,\delta) \gg\!= \Big(\lambda y.\mathbf{fold}\ (\lambda\mu, r.\mu \gg\!= \Big(\lambda x.(\lambda\{r'\}.x\cup\{(r,r')\})_*\llbracket\ \mathbf{GENERATE\ UNDER}\ m\ \mathbf{LIMIT}\ 1\rrbracket(\gamma,\delta[\textsc{id}\mapsto r]).\text{meas}\Big)\ \delta_{\{\}}\ y\Big)$$

(f) **Semantics of rowModel Expressions**

$$\llbracket\textsc{id} : M[?\textsc{id}]\{\textsc{cols}\}\rrbracket(\gamma,\delta) = (\gamma(\textsc{id}).\text{meas}, \gamma(\textsc{id}).\text{pdf}) \qquad \llbracket\ \mathbf{RENAME}\ m\ \mathbf{AS}\ \textsc{id}'\rrbracket(\gamma,\delta) = \llbracket m\rrbracket(\gamma,\delta)$$

$$\llbracket m\ \mathbf{GIVEN}\ c^1 : C^1\{\textsc{cols}\}\rrbracket(\gamma,\delta) = \mathbf{Cond}(\llbracket m\rrbracket(\gamma,\delta), \llbracket c^1\rrbracket(\gamma,\delta[\textsc{id}\mapsto\llbracket m\rrbracket(\gamma,\delta)]))$$

$$\llbracket\textsc{id}\ \mathbf{GIVEN}\ c^0 : C^0\{\textsc{cols}'\}\rrbracket(\gamma,\delta) = \mathbf{let}\ (\pi, v) = \llbracket c^0\rrbracket((\gamma,\delta[\textsc{id}\mapsto\llbracket\textsc{id}\rrbracket(\gamma,\delta)]))\ \mathbf{in}\ \mathbf{Dis}(\gamma(\textsc{id}),\pi,v)$$

(g) **Semantics of Event Expressions**

$$\llbracket\textstyle\bigwedge_{1\le i\le 2} c_i^1\rrbracket(\gamma,\delta) = \textstyle\bigcap_{1\le i\le 2}\llbracket c_i^1\rrbracket(\gamma,\delta)$$

$$\llbracket\textstyle\bigvee_{1\le i\le 2} c_i^1\rrbracket(\gamma,\delta) = \textstyle\bigcup_{1\le i\le 2}\llbracket c_i^1\rrbracket(\gamma,\delta)$$

$$\llbracket\textsc{id}.\textsc{col}_i\ op\ e : C^1\{\textsc{cols}\}\rrbracket(\gamma,\delta) = \{(x_1,\ldots,x_n)\in\llbracket\textsc{cols}\rrbracket \mid x_i\ op_l\ \llbracket e\rrbracket(\gamma,\delta)\}$$

(h) **Semantics of Event-0 Expressions**

$$\llbracket\bigwedge_{1\le i\le 2} c_i^0\rrbracket(\gamma,\delta) = \begin{cases}\mathbf{let}\ {}_{1\le i\le 2}(f_i, v_i) = \llbracket c_i^0\rrbracket(\gamma,\delta)\ \mathbf{in}\\ (\lambda x.(f_1(x), f_2(x)), (v_1, v_2))\end{cases}$$

$$\llbracket\textsc{id}.\textsc{col}_i\ =\ e\rrbracket(\gamma,\delta) = (\pi_i, \llbracket e\rrbracket(\gamma,\delta))$$

Fig. 5. Denotational semantics of GenSQL.

Event-0 expressions (Fig. 5h). $\llbracket c^0 : C^0\{\textsc{cols}\}\rrbracket(\gamma,\delta)$ is interpreted as a pair of a projection function π and a value v in the codomain of the projection. v is used to specify the point at which we want to condition or evaluate a density, and π is used to project the model to the relevant subspace, which we detail in the paragraph on rowModel expressions.

Table expressions (Fig. 5e). We interpret closed tables expressions $\vdash t : T[?\textsc{id}]\{\textsc{cols}\}$ as measures on their columns, i.e. elements of $\mathcal{P}\Big(\mathsf{Tab}\llbracket T[?\textsc{id}]\{\textsc{cols}\}\rrbracket\Big)$. We write $\mu \gg\!= \kappa$ for the composition of a measure μ on X with a kernel $X \to \mathcal{P}Y$, defined by $\mu \gg\!= \kappa(dy) = \int \kappa(x, dy)\mu(dx)$. Given a measurable function $f : X \to Y$, we denote the pushforward measure by $f_*\mu(A) := \mu(f^{-1}(A))$.

We use functional programming notation for the mathematical functions **filter** , **map** , **map2** , **fold** . Given a bag S and a function $f : S \to \mathbb{B}$, we define the bag **filter** $f\ S := \{x \in S \mid f(x) \neq 0\}$. Likewise, we define **map** $f\ S := \{f(x) \mid x \in S\}$ and **map2** $f\ S\ T := \{f(x, y) \mid x \in S, y \in T\}$. A function $f : X \times Y \to Y$ is commutative [21] if $f(x_1, f(x_2, y)) = f(x_2, f(x_1, y))$ for all x_1, x_2, y. Given a commutative function $f : X \times Y \to Y$, we further define **fold** $f : Y \times \mathbf{Bag}(X) \to Y$ by **fold** $f\ y_0\{x_1, \ldots, x_n\} = f(x_1, f(x_2, \ldots f(x_n, y_0) \ldots))$.

RowModel expressions (Fig. 5f). The semantics of rowModels is more involved, as conditioning statements GIVEN c^0 require conditioning on events of probability 0. We first review the minimal setting that helps us define conditioning on event-0 expressions. Given measurable spaces A, B with reference measure ν_A, ν_B, a measure μ on $A \times B$ admits an (A, B) *disintegration* if we can write $\mu = \nu_A \otimes \kappa$ for some measure kernel κ such that for all $a \in A$, $\kappa(a)$ has a density $p(- \mid a)$ w.r.t. ν_B. A *valid decomposition* (A, B) for $\prod_{1 \leq i \leq n} [\![\sigma_i]\!]$ is given by $A = \prod_{j \in J} [\![\sigma_j]\!]$ for some $J \subseteq \{1, \ldots, n\}$ and $B = \prod_{j \in \{1,\ldots,n\} - J} [\![\sigma_j]\!]$. A measure $\mu \in \mathcal{P}(\prod_{1 \leq i \leq n} [\![\sigma_i]\!])$ is *admissible* if it admits an (A, B) disintegration for all valid decompositions (A, B) of $\prod_{1 \leq i \leq n} [\![\sigma_i]\!]$.

We consider measures μ on spaces X with chosen disintegrations and (marginal) densities w.r.t. the reference measure. More precisely, we interpret a rowModel ID from the global context Γ as a quadruple $\text{Tab}[\![\text{ID}]\!] := (\mu, p, \{\kappa_A\}_A, \{p\}_A)$. Here, μ is a measure denoting the unconditioned model, and p a density of μ w.r.t. the reference measure. For each valid decomposition (A, B) of the columns of ID, the kernel κ_A is an (A, B)-disintegration of μ. For all $a \in A$, $p_A(- \mid a)$ is a density for $\kappa_A(a)$ w.r.t. the reference measure ν_B. If v is a partial assignment of the variables in B, we also write $p_A(v \mid a)$ for the marginal density of $\kappa_A(a)$ at v obtained from $p_A(- \mid a)$ by integrating out the missing variables in v. We denote by $\mathcal{P}_{\text{adm}}(X)$ the set of such quadruples $(\mu, p, \{\kappa_A\}_A, \{p\}_A)$, where μ is a measure on X. Given $m \in \mathcal{P}_{\text{adm}}(X)$, we write m.meas for its first component μ, m.pdf for the density p, $m.A$ for the kernel κ_A, and $m.A$.pdf for the density p_A. Using this notation, given an event-0 c^0 denoting a projection π and value v, the expression $m.\text{pdf}(v)$ gives a marginal density of the model m at v; i.e. $m.\text{pdf}(v)$ is a version of the density of $\pi_* m.\text{meas}$ evaluated at v. We assume that all the models in the context are admissible, which is enforced in the semantics of contexts.

The models used in queries are built from admissible models and will carry chosen densities, which is enforced in the semantics of rowModel expressions. We write $\mathcal{P}_{\text{dens}}(X)$ for the set of pairs (μ, p) where μ is a measure on $X := X_1 \times \ldots \times X_n$ and p is either a density of μ w.r.t. the reference measure, or of the form $\lambda(x_1, \ldots, x_n).q(x_{i_1}, \ldots, x_{i_k})$ for some $i_1, \ldots, i_k$, and where q is a marginal density of μ on $X_{i_1} \times \ldots \times X_{i_k}$ w.r.t. the reference measure. The second case is used to represent the density of a model conditioned on an event-0 expression.

Conditioning on events-0 requires access to a disintegration of the model at the point v, which is possible thanks to the restriction from the type-system. For $m \in \mathcal{P}_{\text{adm}}(X)$, $\pi : X \to Y$ a projection function, and $v \in Y$, we define $\mathbf{Dis}(m, \pi, v) := (m.\pi(X)(v) \otimes \delta_v, m.\pi(X).\text{pdf}(v))$.

For conditioning on events, given $m \in \mathcal{P}_{\text{dens}}(X)$ and a measurable $S \subseteq X$, we define

$$\mathbf{cond}(m, S) := \begin{cases} \left(\lambda S'.\dfrac{m.\text{meas}(S \cap S')}{m.\text{meas}(S)}, \lambda x.\dfrac{\mathbb{1}_S(x) m.\text{pdf}(x)}{m.\text{meas}(S)}\right) & \text{if } m.\text{meas}(S) > 0 \\ (\delta_{\{\star,\ldots,\star\}}, \mathbb{1}_{\{(\star,\ldots,\star)\}}) & \text{otherwise} \end{cases}$$

4 ABSTRACT MODEL INTERFACE AND QUERY PLANNER

This section presents a query planner that automatically lowers GenSQL queries to programs that operate on tables and rowModels. This lowering depends on the Abstract rowModel Interface (AMI) which we assume all loaded rowModels must satisfy. The AMI is a flexible interface that many rowModel implementations can easily satisfy. This flexibility means that model implementations can strike different expressiveness-speed-accuracy trade-offs, and give different guarantees.

Appendix A.2 compares an exact SPPL backend to an approximate Gen.clj backend on 5 queries.

In what follows, we define the AMI and show how to lower GenSQL queries to programs that access rowModels through the AMI interface. We showcase the flexibility of the AMI by proving formal guarantees for two different implementations of the AMI. We show in Section 4.3 that if the AMI is implemented in a PPL with exact inference, then GenSQL queries can be lowered to programs in a semantics-preserving way. We then show in Section 4.4 that if the AMI is implemented in a PPL with approximate inference, then GenSQL queries can be lowered to programs that encode asymptotically sound estimators for PROBABILITY OF expressions and asymptotically sound samplers for GENERATE UNDER expressions.

4.1 Abstract Model Interface (AMI)

A rowModel represents a probability distribution on rows with a fixed set of columns. The AMI captures the intuition that a model should be able to produce samples and compute probabilities and densities for all conditioned versions of the distribution it represents. For each rowModel $M[?\text{ID}]\{\text{COLS}\}$, the AMI requires the existence of the following three methods:

$$\begin{aligned}
\textbf{simulate}_{\text{ID}} &: (C^0\{\text{COLS}'\}, C^1\{\text{COLS}\}) \to T[?\text{ID}]\{\text{COLS}\}\\
\textbf{logpdf}_{\text{ID}} &: (C^0\{\text{COLS}'\}, C^1\{\text{COLS}\}, C^0\{\text{COLS}''\}) \to \textbf{Real}\\
\textbf{prob}_{\text{ID}} &: (C^0\{\text{COLS}'\}, C^1\{\text{COLS}\}, C^1\{\text{COLS}\}) \to \textbf{Ranged}(0,1).
\end{aligned}$$

where $\text{COLS}', \text{COLS}'' \subseteq \text{COLS}$. These methods should behave as follows:

- $\textbf{simulate}_{\text{ID}}(c^0, c^1)$ returns a sample from a model with identifier ID, conditioned on the event-0 c^0 and event c^1.
- $\textbf{logpdf}_{\text{ID}}(c^0, c^1, c_2^0)$ returns the (marginal if $\text{COLS}'' \subsetneq \text{COLS}$) log-density of the model ID conditioned on the event-0 c^0 and event c^1, at the point c_2^0.
- $\textbf{prob}_{\text{ID}}(c^0, c^1, c_2^1)$ returns the probability of the event c_2^1 under the model ID, conditioned on the event-0 c^0 and event c^1.

A non-conditioned model is recovered by letting the subset COLS' to be empty. The precise usage of these methods is given in the next section. The AMI methods can have different formal semantics, capturing different aspects of the backend probabilistic model it abstracts. These semantics reflect different implementation strategies implementing conditional sampling and probability evaluation. Appendix C shows how different model classes can implement the AMI. In particular, we show that SPPL [73] and truncated multivariate Gaussians satisfy the exact AMI, and that any PPL implementing ancestral sampling will satisfy the approximate AMI. We next describe how the GenSQL query planner lowers queries to programs that rely on the AMI, before giving details about the semantics and correctness guarantees.

4.2 Lowering GenSQL to Queries on the AMI

The lowering procedure from GenSQL to a lowered language is given in two steps: (i) a normalization transform for GenSQL queries; and (ii) a program transform to the lowered language.

Normalization of GenSQL Queries. The normalization (see Appendix D.1) simply simplifies RENAME statements and aggregates events in a single conditioning statement. It leads to the following normal forms, where GIVEN clauses are optional:

- Probability queries: PROBABILITY OF c_1^i UNDER (ID GIVEN c^0 GIVEN c^1).
- Generate queries: GENERATE UNDER (ID GIVEN c^0 GIVEN c^1) LIMIT e and t GENERATIVE JOIN (ID GIVEN c^0 GIVEN c^1).

$$\text{base type } \sigma ::= \sigma_c \mid \sigma_d \quad \text{ground type } \sigma_g ::= \sigma \mid (\sigma_1, \ldots, \sigma_n) \quad \text{event type } \mathcal{E} ::= C^1[\sigma_g] \mid C^0[\sigma_g]$$

$$\text{type } \tau ::= \mathbf{Bag}[\sigma_g] \quad \text{operator } op ::= + \mid - \mid \times \mid \div \mid \wedge \mid \vee \mid = \quad \text{rowModel } \mathcal{M} ::= M[\sigma_g]$$

$$\text{primitives } f ::= \mathbf{mapreduce} \mid \mathbf{map} \mid \mathbf{filter} \mid \mathbf{replicate} \mid \mathbf{join} \mid \mathbf{exp} \mid \mathbf{singleton} \mid \mathbf{simulate}_{\text{ID}} \mid \mathbf{prob}_{\text{ID}} \mid \mathbf{logpdf}_{\text{ID}}$$

$$\text{term } t ::= c \mid \text{ID} \mid f(t_1, \ldots, t_n) \mid x \mid (t_1, \ldots, t_n) \mid \pi_i\, t \mid t_1\; op\; t_2$$

$$\frac{\Gamma \vdash t_1 : C^0[\sigma_g^1] \quad \Gamma \vdash t_2 : C^0[\sigma_g^2]}{\Gamma \vdash t_1 \wedge t_2 : C^0[\sigma_g^1, \sigma_g^2]}$$

$$\frac{\Gamma \vdash t_1 : C^1[\sigma_g] \quad \Gamma \vdash t_2 : C^1[\sigma_g] \quad op \in \{\wedge, \vee\}}{\Gamma \vdash t_1\; op\; t_2 : C^1[\sigma_g]} \qquad \frac{\Gamma \vdash t : \sigma_i \quad op \in \{=, <, >\} \quad (\sigma_i, op) \neq (\sigma_c, =)}{\Gamma, \text{ID} : M[(\sigma_1, \ldots, \sigma_n)] \vdash (\text{ID}, i)\; op\; t : C^1[(\sigma_1, \ldots, \sigma_n)]}$$

$$\frac{\Gamma, \text{ID} : M[\sigma_g] \vdash c^i : C^i[\sigma_g] \quad \Gamma, \text{ID} : M[\sigma_g] \vdash c_1^1 : C^1[\sigma_g]}{\Gamma, \text{ID} : M[\sigma_g] \vdash \mathbf{prob}_{\text{ID}}(c^0, c^1, c_1^1,) : \mathbf{Ranged}(0,1)} \qquad \frac{\Gamma, \text{ID} : M[\sigma_g] \vdash c^i : C^i[(\sigma_1, \ldots, \sigma_n)]}{\Gamma, \text{ID} : M[\sigma_g] \vdash \mathbf{simulate}_{\text{ID}}(c^0, c^1) : \mathbf{Bag}[\sigma_g]}$$

$$\frac{\Gamma \vdash t : \sigma_i}{\Gamma, \text{ID} : M[(\sigma_1, \ldots, \sigma_n)] \vdash (\text{ID}, i) = t : C^0[\sigma_i]} \qquad \frac{\Gamma, \text{ID} : M[\sigma_g] \vdash c^i : C^i[\sigma_g] \quad \Gamma, \text{ID} : M[\sigma_g] \vdash c_1^0 : C^0[\sigma_g]}{\Gamma, \text{ID} : M[\sigma_g] \vdash \mathbf{logpdf}_{\text{ID}}(c^0, c^1, c_1^0) : \mathbf{Real}}$$

Fig. 6. A selected subset of the syntax and type system of the lowered language.

Lowering Language (Fig. 6). It is a first-order simply-typed lambda calculus with second-order operations acting on bags, and primitives for the AMI. It also contains a version of events and events-0 which can be used by AMI primitives. Operations like **map** , **filter** and **exp** have their usual meaning, and their typing along with those for constants, tuples, projections, and arithmetic operations are standard and recalled in Appendix D (Fig. 20). **join** takes two bags of tuples and returns their Cartesian product. **replicate** evaluates its bag argument n times and returns the union of all the resulting bags. **mapreduce** takes a bag of tuples and a function f from tuples to bags, and returns the union of all the bags obtained by applying f to each tuple in the input bag.

Lowering program transform (Fig. 7). After obtaining a normal form query, the planner applies a program transformation $\mathcal{T}_\delta\{\cdot\}$ from normalized GenSQL queries to the lowered language, defined by pattern matching on the structure of the query. It carries a local context δ of variables (a finite map from identifiers to variable names) which are bound in the surrounding program. Similarly to the local context Δ in GenSQL, δ will start empty $[]$ at the root of the syntax tree. It is used to rename variables in the lowered query. The rationale is that a table identifier ID in Δ will be transformed to a variable r representing a tuple being iterated over by a **map** or **fold** primitive. A rowModel identifier ID, on the other hand, will be uniquely accessible and identified from the global context Γ, thanks to the normalization procedure which ensures that no rowModel is renamed in the normalized query. A simple proof by induction shows that the transformation preserves typing.

Proposition 4.1. *If* $\Gamma, [] \vdash t : T[?\mathit{ID}]\{\mathit{COLS}\}$, *then* $\mathcal{T}\{\Gamma\} \vdash \mathcal{T}_{[]}\{t\} : \mathcal{T}\{T[?\mathit{ID}]\{\mathit{COLS}\}\}$.

4.3 Lowering Guarantees for Exact Backend

A large class of models supports exact inference, e.g. those expressible in SPPL [73] and truncated multivariate Gaussians. These models satisfy the exact AMI and are able to return exact samples from simulate, and compute exact marginal logpdf and prob queries, even for conditioned models. We make this precise by giving a measure semantics on the lowered language (Fig. 21) and show that the program transform $\mathcal{T}\{\cdot\}$ preserves the semantics of the lowered query (Theorem 4.2). In particular, all the scalar computations in the query are deterministic and that the generated synthetic data comes from exact conditional distributions.

The denotational semantics (Appendix D, Fig. 21) of the lowered language is mostly standard and resembles the measure-theoretic semantics of GenSQL given in Fig. 5. Terms $\Gamma \vdash e : \sigma_g$ are

(a) **Translating Types and Contexts**

$$\mathcal{T}\{T[?\text{ID}]\{\text{COL}_1:\sigma_1,\ldots,\text{COL}_n:\sigma_n\}\} = \textbf{Bag}[(\sigma_1,\ldots,\sigma_n)]$$
$$\mathcal{T}\{M[?\text{ID}]\{\text{COL}_1:\sigma_1,\ldots,\text{COL}_n:\sigma_n\}\} = M[(\sigma_1,\ldots,\sigma_n)]$$
$$\mathcal{T}\left\{C^i\{\text{COL}_1:\sigma_1,\ldots,\text{COL}_n:\sigma_n\}\right\} = C^i[(\sigma_1,\ldots,\sigma_n)]$$
$$\mathcal{T}\{\Gamma, T[?\text{ID}]\{\text{COLS}\}\} = \mathcal{T}\{\Gamma\}, \text{ID} : \mathcal{T}\{T[?\text{ID}]\{\text{COLS}\}\}$$
$$\mathcal{T}\{\Gamma, M[?\text{ID}]\{\text{COLS}\}\} = \mathcal{T}\{\Gamma\}, \text{ID} : \mathcal{T}\{M[?\text{ID}]\{\text{COLS}\}\}$$
$$\mathcal{T}\{\sigma\} = \sigma \quad \mathcal{T}\{[\,]\} = [\,]$$

(b) **Translating Event and Event-0 Expressions**

$$\mathcal{T}_\delta\{\text{ID.COL}_i = e\} = (\text{ID}, i) = \mathcal{T}_\delta\{e\}$$
$$\mathcal{T}_\delta\{\text{ID.COL}_i \ > \ e\} = (\text{ID}, i) > \mathcal{T}_\delta\{e\}$$
$$\mathcal{T}_\delta\{\text{ID.COL}_i \ < \ e\} = (\text{ID}, i) < \mathcal{T}_\delta\{e\}$$
$$\mathcal{T}_\delta\{c_1 \wedge c_2\} = \mathcal{T}_\delta\{c_1\} \wedge \mathcal{T}_\delta\{c_2\}$$
$$\mathcal{T}_\delta\{c_1 \vee c_2\} = \mathcal{T}_\delta\{c_1\} \vee \mathcal{T}_\delta\{c_2\}$$

(c) **Translating RowModel Queries**

$$\mathcal{T}_\delta\left\{\textbf{PROBABILITY OF } c_2^0 \textbf{ UNDER } \text{ID} \textbf{ GIVEN } c^0 \textbf{ GIVEN } c^1\right\} = \textbf{exp}\left(\textbf{logpdf}_{\text{ID}}(\mathcal{T}_\delta\{c^0\}, \mathcal{T}_\delta\{c^1\}, \mathcal{T}_\delta\{c_2^0\})\right)$$
$$\mathcal{T}_\delta\left\{\textbf{PROBABILITY OF } c_2^1 \textbf{ UNDER } \text{ID} \textbf{ GIVEN } c^0 \textbf{ GIVEN } c^1\right\} = \textbf{prob}_{\text{ID}}(\mathcal{T}_\delta\{c^0\}, \mathcal{T}_\delta\{c^1\}, \mathcal{T}_\delta\{c_2^1\},)$$
$$\mathcal{T}_\delta\left\{\textbf{GENERATE UNDER } \text{ID} \textbf{ GIVEN } c^0 \textbf{ GIVEN } c^1 \textbf{ LIMIT } e\right\} = \textbf{replicate}\,(\mathcal{T}_\delta\{e\}, \textbf{simulate}_{\text{ID}}(\mathcal{T}_\delta\{c^0\}, \mathcal{T}_\delta\{c^1\}))$$
$$\mathcal{T}_\delta\left\{t : T[\text{ID}']\{\text{COLS}\} \textbf{ GENERATIVE JOIN } \text{ID} \textbf{ GIVEN } c^0 \textbf{ GIVEN } c^1\right\} =$$
$$\textbf{mapreduce}\ \left(\lambda r.\textbf{join}\left(\textbf{singleton}(r), \textbf{simulate}_{\text{ID}}(\mathcal{T}_{\delta[\text{ID}'\mapsto r]}\{c^0\}, \mathcal{T}_{\delta[\text{ID}'\mapsto r]}\{c^1\})\right), \mathcal{T}_\delta\{t\}\right)$$

(d) **Translating Scalar Expressions**

$$\mathcal{T}_\delta\{c\} = c \quad \mathcal{T}_{\delta[\text{ID}\mapsto r]}\{\text{ID.COL}_i\} = \pi_i(r) \quad \mathcal{T}_\delta\{op(e_1,\ldots,e_n)\} = op(\mathcal{T}_\delta\{e_1\},\ldots\mathcal{T}_\delta\{e_n\})$$

(e) **Translating Table Expressions**

$$\mathcal{T}_\delta\{\textbf{ RENAME } t \textbf{ AS } \text{ID}\} = \mathcal{T}_\delta\{t\};\ \mathcal{T}_\delta\{\text{ID}\} = \text{ID}$$
$$\mathcal{T}_\delta\{t_1 \textbf{ JOIN } t_2\} = \textbf{join}(\mathcal{T}_\delta\{t_1\}, \mathcal{T}_\delta\{t_1\})$$
$$\mathcal{T}_\delta\{t : T[\text{ID}]\{\text{COLS}\} \textbf{ WHERE } e\} = \textbf{filter}\ (\lambda r.\mathcal{T}_{\delta[\text{ID}\mapsto r]}\{e\}, \mathcal{T}_\delta\{t\})$$
$$\mathcal{T}_\delta\left\{\begin{array}{l}\textbf{SELECT } \overline{e} \textbf{ AS } \overline{\text{COL}} \\ \textbf{FROM } t : T[\text{ID}]\{\text{COLS}\}\end{array}\right\} = \textbf{map}\ (\lambda r.\mathcal{T}_{\delta[\text{ID}\mapsto r]}\{\overline{e}\}, \mathcal{T}_\delta\{t\})$$

Fig. 7. The lowering transformation $\mathcal{T}\{\cdot\}$.

interpreted as deterministic measurable functions $[\![\Gamma]\!]_{\text{exact}} \to [\![\sigma_g]\!]_{\text{exact}}$. Terms $\Gamma \vdash e : \textbf{Bag}[\sigma_g]$ are interpreted as probability kernels $[\![\Gamma]\!]_{\text{exact}} \to \mathcal{P}\textbf{Bag}([\![\sigma_g]\!]_{\text{exact}})$, where substitution for these programs is interpreted using the Kleisli composition for the point process monad [21]. By induction on the structure of GenSQL programs t in context $\Gamma;\Delta$, we can show (proof in Appendix D.3):

THEOREM 4.2 (EXACT AMI GUARANTEE). *Let $\Gamma, [\,] \vdash t : T[?\mathit{ID}]\{\mathit{COLS}\}$. Then, for every evaluation of the context γ,*

$$[\![t]\!](\gamma, [\,]) = [\![\mathcal{T}_{[\,]}\{t\}]\!]_{exact}(\gamma).$$

4.4 Approximate Backend Guarantee

By relying on approximate probabilistic inference, general-purpose PPLs can express large classes of models in which exact inference is intractable. In addition, programmable inference [51] ensures PPLs can support a diverse class of probabilistic models without sacrificing inference quality. We give a new denotational semantics for the lowered language that is appropriate for reasoning in scenarios where the rowModels are implemented in PPLs with approximate Monte Carlo inference.

Monte Carlo algorithms are typically parameterized by a positive integer n specifying a compute budget, such as the number of particles in a sequential Monte Carlo (SMC) algorithm [18] or the number of samples in a Markov Chain Monte Carlo (MCMC) algorithm [64]. The algorithm specifies a sequence of distributions or estimators that converge in some sense to a quantity of interest as $n \to \infty$. In the case of approximate sampling algorithms, most typically the distribution of the generated samples converges weakly to the target distribution, and in the case of parameter estimation the algorithm produces a strongly consistent estimator of the target parameter [18, 64].

Random variable semantics. Our denotational semantics for approximate AMI implementations is motivated by the above discussion. We assume the existence of an ambient probability space $(\Omega, \mathcal{F}, \mathbb{P})$ and associate with each term a sequence of random variables approximating the term in the exact semantics. As an example, the approximate semantics of $[\![\mathbf{map}\ (x.t_1)\ t_2]\!]_{\text{approx}}$ in the context γ and at the "random seed" $\omega \in \Omega$ is given at the n-th approximation by

$$[\![t_2]\!]_{\text{approx}}(\gamma, \omega)_n \gg\!= \left(\lambda S.\mathbf{return}\ \left\{\lambda x'.[\![t_1]\!]_{\text{approx}}(\gamma[x \mapsto x'], \omega)_n\ y \,\middle|\, y \in S\right\}\right).$$

This means that we first obtain the n-th approximation of the input t_2, which is a measure on tables, which we then evaluate to obtain a concrete table, S. We then apply the function to each row obtained by the n-th approximation of t_1. The full semantics is given in Appendix D.4, Fig. 23. We assume the following hold:

- For each rowModel identifier $\textsc{id} : M[(\sigma_1, \ldots, \sigma_k)]$ in environment γ, event $c^1 : C^1[(\sigma_1, \ldots, \sigma_k)]$, and event-0 $c^0 : C^0[(\sigma_1, \ldots, \sigma_k)]$, there exists a sequence of probability measures $\{\mu^n_{\textsc{id};[\![c^0]\!]_{\text{approx}}(\gamma)_n, [\![c^1]\!]_{\text{approx}}(\gamma)_n}\}$ on $\mathbf{Bag}\ \prod_{i=1}^{k}[\![\sigma_i]\!]$;
- for $\textsc{id}$, γ, c^1 and c^0 as above, and $c^1_2 : C^1[(\sigma_1, \ldots, \sigma_k)]$, there exists a sequence of real random variables $\{P^n_{\textsc{id};[\![c^0]\!]_{\text{approx}}(\gamma)_n, [\![c^1]\!]_{\text{approx}}(\gamma)_n, [\![c^1_2]\!]_{\text{approx}}(\gamma)_n}\}$ which takes values in $[0, 1]$ $\mathbb{P}$-almost surely;
- for $\textsc{id}$, γ, c^1 and c^0 as above, and $c^0_2 : C^0[(\sigma_1, \ldots, \sigma_k)]$, there exists a sequence of real random variables $\{L^n_{\textsc{id};[\![c^0]\!]_{\text{approx}}(\gamma)_n, [\![c^1]\!]_{\text{approx}}(\gamma)_n, [\![c^0_2]\!]_{\text{approx}}(\gamma)_n}\}$.

These random sequences represent the sequences of approximations produced by the implementation of the AMI. In general, for a given term t the convergence of sequences associated with its sub-terms do not imply that the sequence associated with t converges. For instance, consider evaluating the following query in an appropriate context (γ, δ):

$$\textsc{select id.col from id where id.col} \leq (\ \textsc{probability of id}'.\textsc{col}' = 7).$$

If the value of the term $\textsc{probability of id}'.\textsc{col}' = 7$ is approximated, even if we can make this approximation arbitrarily accurate, the output of the query need not converge. For example, if the table $\textsc{id}$ contains a row in which the value of $\textsc{col}$ is exactly $[\![\ \textsc{probability of id}'.\textsc{col}' = 7]\!](\gamma, \delta)$ but the approximation converges to the true value from below, this row will not be included in the query result no matter the accuracy of the approximation. Intuitively, this arises from the fact that the indicator functions of half intervals are not continuous.

In order for the lowered queries to denote asymptotically sound estimators for the original queries, we require that the implementation of the AMI methods are asymptotically sound, and write $\lim_n \gamma_n$ to denote an evaluation of the context γ in which each random variable is replaced by its limit as $n \longrightarrow \infty$. In Appendix D.4, we formalize the notions of *safe* queries and asymptotically *sound* AMI implementations and details of the proofs. We then give the following guarantee.

Theorem 4.3 (Consistent AMI Guarantee). *Let* $\Gamma, [] \vdash t : T[?\textsc{id}]\{\textsc{cols}\}$ *be a safe query and suppose the AMI methods have asymptotically sound implementations. Then, for every evaluation of the context* γ, $\mathbb{P}$*-almost surely*

$$\lim_n([\![\mathcal{T}_{[]}\ \{t\}]\!]_{approx})(\gamma) = [\![t]\!](\lim_n \gamma, []).$$

5 EVALUATION

The performance of an open-source Clojure implementation of GenSQL is evaluated against other systems that have similar capabilities. We test runtime, the effect of optimizations, and runtime overhead of our system over alternative implementations of the same task. Experiments were run on an Amazon EC2 C6a instance with Ubuntu 22.04, 4 vCPUs and 8.0 GiB RAM.

The probabilistic models used in the evaluation are obtained using probabilistic program synthesis [74, Chapter 3]. Each model is an ensemble of "MultiMixture" probabilistic programs [69, Section 6], which are posterior samples from the CrossCat model class [50], generated using ClojureCat [16]. An ensemble of 10 probabilistic programs is used in Section 5.1 and 12 programs in Section 5.2.

Table 1. Runtime (sec) comparison of GenSQL and BayesDB [52] on 10 benchmark queries (Appendix F) for evaluating probability densities of measure-zero events.

	GenSQL (ClojureCat Backend)	BayesDB (CGPM Backend)	Speedup
Q1	0.24 ± 0.03	0.59 ± 0.16	2.5x
Q2	0.29 ± 0.03	1.15 ± 0.2	4.0x
Q3	0.43 ± 0.06	1.72 ± 0.28	4.0x
Q4	0.48 ± 0.06	2.25 ± 0.27	4.7x
Q5	0.57 ± 0.07	2.68 ± 0.36	4.7x
Q6	0.33 ± 0.06	0.55 ± 0.23	1.7x
Q7	0.49 ± 0.05	1.53 ± 0.26	3.1x
Q8	0.46 ± 0.03	1.81 ± 0.21	3.9x
Q9	0.37 ± 0.03	2.51 ± 0.32	6.8x
Q10	0.45 ± 0.04	2.87 ± 0.39	6.4x
Mean	0.41 ± 0.11	1.77 ± 0.83	4.3x

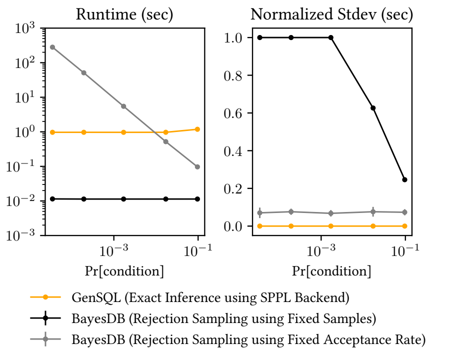


Fig. 8. Runtime/stdev comparison of GenSQL and BayesDB [52] on 5 benchmark queries for evaluating probabilities of positive measure events.

5.1 Performance and Usability

Runtime comparison. Table 1 shows a comparison of the runtime to solve 10 benchmark queries (Appendix F) adapted from Charchut [16, Tables 4.2 and 4.3] using GenSQL (with the ClojureCat backend) and BayesDB (with the CGPM backend [66]) for evaluating exact probability densities. Section 5 compares the runtime and standard deviation for computing the probabilities of positive measure events. GenSQL (with the SPPL backend [73]) delivers exact solutions, whereas BayesDB delivers approximate solutions using rejection sampling. Two rejection strategies in BayesDB are shown in Section 5: a fixed number of samples (faster but higher variance) or a fixed acceptance rate (slower but lower variance), which both are inferior to exact solutions from GenSQL.

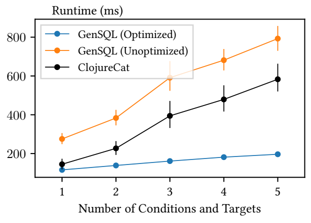
The performance gains in GenSQL are due to three main reasons: the ClojureCat backend is faster than the CGPM backend in BayesDB, GenSQL has optimizations (discussed below) that exploit repetitive computations, and GenSQL itself is implemented in Clojure, a performant language.

Optimizations and system overhead. GenSQL leverages two classes of optimizations: caching (of the likelihood queries and conditioned models) and exploiting independence relations between columns. The latter allows us to simplify a query such as **PROBABILITY OF** ID.$x > 42$ **UNDER** ID **GIVEN** ID.$y = 17$ to the semantically equivalent query **PROBABILITY OF** ID.$x > 42$ **UNDER** ID if the columns x and y are independent. Appendix B gives a detailed account of the optimizations.

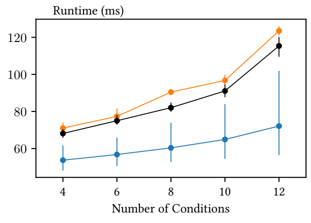
In Fig. 9, the unoptimized GenSQL queries have a 1.1-1.6x overhead compared to the pure ClojureCat baseline. The optimizations reduce the overhead and can sometimes drastically improve performance. In addition, caching seems to heavily reduce the variance in the runtime of the queries. In Fig. 9b, the effect of the independence optimization varies between replicates, as these are different CrossCat model samples, which explains the higher variance in query runtime.

Code comparison. Figure 10 compares the code required in GenSQL, pure Python using SPPL [73], and pure Clojure using ClojureCat [16], for a conditional probability query. Figure 10a shows how GenSQL gains clarity by specializing in data that comes from database tables. In contrast, both SPPL and ClojureCat require users to hand-write the looping/mapping over the data, which is error prone. For instance, the code in Fig. 10c will crash if the table has missing values. In Fig. 10b, ClojureCat requires conditions to be maps. Users can decide if they encode columns with strings, symbols, or keywords. If this choice does not align with the key type returned by the CSV reader, the query will run but conditioning will result in a null-op.

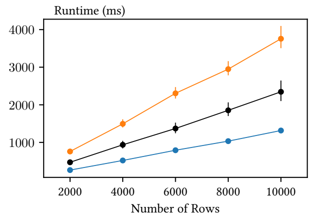
In Appendix A.1, we compare a single line query on a conditioned model in GenSQL to the equivalent code in Scikit-learn [60] on the Iris data from the UCI ML repository. The model querying



(a) Varying number of conditions and targets in the **PROBABILITY OF** queries shown in Table 1.



(b) Varying number of conditions in **GIVEN** clause for **GENERATE UNDER** queries (caching does not apply).



(c) Varying number of rows in a data table used in the **FROM** clause of **SELECT** with a **PROBABILITY OF** query.

Fig. 9. Runtime comparison between GenSQL (ClojureCat backend) and raw ClojureCat [16].

```
SELECT PROBABILITY OF
    Period_minutes = 98.6,
    Type_of_Orbit = "Sun-Synchro.",
    Contractor = "Lockheed Martin"
  UNDER model GIVEN Country_of_Operator, Launch_Mass_kg
FROM data
```

(a) GenSQL

```
(let
  [event {:Period_minutes 98.6
          :Type_of_Orbit "Sun-Synchro."
          :Contractor "Lockheed Martin"}
   cols-in-condition [:Country_of_Operator,
                      :Launch_Mass_kg]]
  (->> data
       (map #(select-keys cols-in-condition %))
       (mapv #(exp (gpm/logpdf model event %)))))
```

(b) ClojureCat (Clojure)

```
event = {
    "Period_minutes": 98.6,
    "Type_of_Orbit": "Sun-Synchro.",
    "Contractor": "Lockheed Martin",
}
cols_in_condition = ["Country_of_Operator", "Launch_Mass_kg"]

targets = {sppl.transforms.Identity(k): v for k, v in event.
  items()}
constraints = [
    {
        sppl.transforms.Identity(column): value
        for column, value in row.items()
        if column in cols_in_condition
    }
    for _, row in data.iterrows()
]
print([exp(spe.constrain(constraint).logpdf(targets)) for
  constraint in constraints])
```

(c) SPPL (Python)

Fig. 10. Comparison of GenSQL, ClojureCat [16], and SPPL [73] code for a conditional probability query.

a	b	c
a_0	b_0	c_0
a_1	b_1	c_1
a_0	b_0	c_0
a_0	b_1	c_0
...	...	...

(a) Table foo

a	b	x	y	z
a_0	b_1	4.2	4.1	0.6
a_1	b_1	-4.4	-5.4	0.2
a_1	b_0	-3.7	-6.2	0.5
a_1	b_1	-6.2	-4.2	0.1
...	...	...	...	...

(b) Table bar to build model

```
SELECT * FROM foo GENERATIVE JOIN bar_model GIVEN *
```

(c) GenSQL

```
# escape into SQLite
ALTER TABLE bar ADD COLUMN c TEXT
UPDATE bar SET c = 'placeholder'
INSERT INTO bar SELECT
  a,b, NULL AS x, NULL AS y, NULL AS z, c
  FROM foo
```

```
INFER a, b, c, x, y, z
FROM bar WHERE rowid > [num rows in foo]
```

(d) BayesDB

Fig. 11. Comparison of GenSQL and BayesDB [52] code. The latter does not support **GENERATIVE JOIN** .

alone in Scikit-learn is more than 50 lines long and clearly error prone, and we find that GenSQL offers a significant advantage in simplicity over such baselines.

Code comparison with BayesDB. Figure 11 shows GenSQL and its closest relative, BayesDB [52], on a **GENERATIVE JOIN** query on synthetic data. The GenSQL code is more concise and simpler than BayesDB's code, which is possible due to the language abstractions for manipulating models. In BayesDB, the user must exit to SQL and hand-code column manipulations to fit the expected fixed pattern to query a model. Section 6 provides a detailed comparison of GenSQL and BayesDB.

5.2 Case Studies on Real World Data

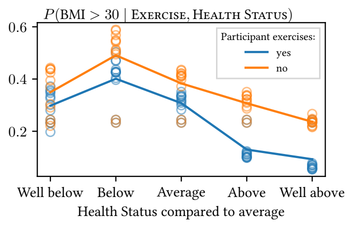
We present two case studies to demonstrate the application of GenSQL to real-world problems: one in medicine (clinical trial data) and one in synthetic biology (wetlab data). The datasets can be costly to obtain and researchers are interested in understanding and analyzing their data.

```
SELECT
  BMI, exercise, health_status, age, smoker
FROM clinical_trial_records
WHERE
  (BMI > 20.3) AND
  (BMI < 38.4) AND
  (PROBABILITY OF BMI UNDER clinical_trial_model GIVEN *) < 0.01
```

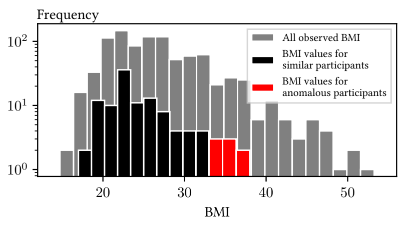
(a) Show participants with anomalous BMI values. Define anomalous as $P(\text{BMI} \mid *) < 0.01$ for all BMI values that are larger than the 5th percentile (20.3) and smaller than the 95th (38.4) in the United states.

BMI	exercise	health_status	age	smoker
38	yes	Above average	40-49	never
33	yes	Well above average	30-39	never
33	yes	Well above average	50-59	never
35	yes	Above average	40-49	never
37	yes	Well above average	40-49	never
33	yes	Well above average	50-59	never
36	yes	Above average	30-39	former
35	yes	Above average	40-49	former

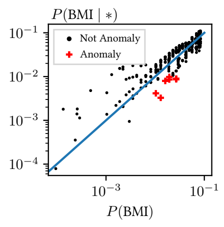
(b) Result from the query in (a). Eight anomalous participants are returned: all are clinically obese but report above-average health status and exercise.



(c) Conditonal probabilites encoded by the underlying model.



(d) Compare anomalous BMI values to normal ones.



(e) Compare conditional and marginal BMI.

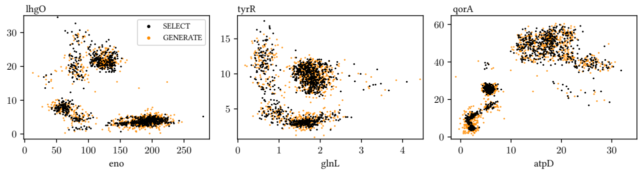
Fig. 12. Case study: Anomaly detection in clinical trials.

In the first case, we show how anomaly detection in GenSQL can be used to check for probable mislabelling of the data. The anomalous rows can also be investigated further to understand the reasons for the anomaly. In the second case, we show how GenSQL can be used to generate accurate synthetic data, capturing the complex relationships between different host genes and experimental conditions. Capturing these relationships with the model helps predict whether a certain experimental condition or modification of the genome has cascading downstream effects through the interrelations between the genes. Such effects can render the cell toxic and kill the bacterium, leading to a failed experiment. The virtual wet lab allows researchers to check for such effects before running costly experiments in the real world.

Anomaly detection in clinical trials. We surface anomalies in data from the (BEAT19) clinical trial [86] which contains data about COVID-19 and records behavior, environment variables, and treatments. Each row represents a clinical trial participant. Figure 12a shows the query and the anomaly criteria used [15]. For each row, it computes the likelihood under the model of obtaining the value BMI given all the other values of the row. The trial participants labeled anomalous (Fig. 12b) all report above-average or well-above-average health and that they exercise, while meeting the World Health Organization's definition of clinicial obesity [82]. Fig. 12d compares the overall population in the trial (grey) with the anomalous individuals (red) and the subset of the overall population that reports the same behavioral covariates (black). The latter means that those individuals have similar values for exercising, health status, age, and smoking habits. For similar individuals, the data suggests a lower BMI, even though much larger BMI values are not infrequent. We can also compare the marginal and the conditional probability of BMI values in the table of clinical trial records (Fig. 12d). The data labeled anomalous (red) is lower than the diagonal line, which highlights the "contextualization" of BMI values that happens by conditioning the models. The goal here is not to detect extreme values, i.e., univariate outliers, which can be done using a WHERE filter. Instead, we are highlighting data points where other cell values in the same row provide a context that renders a data point unlikely. To demonstrate this effect, we first

```
SELECT * FROM gene_expression_data

GENERATE UNDER gene_expression_model
LIMIT 1000
```

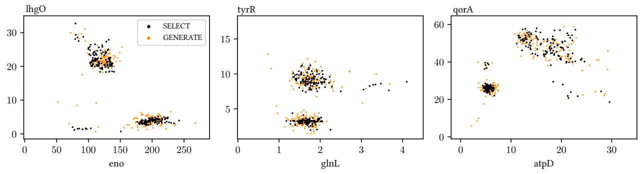
(a) Overall population



(b) 6 genes from the overall population.

```
SELECT * FROM gene_expression_data
WHERE Ara. = "not added" AND IPTG = "added"

GENERATE UNDER gene_expression_model
  GIVEN Ara. = "not added" AND IPTG = "added"
LIMIT 1000
```

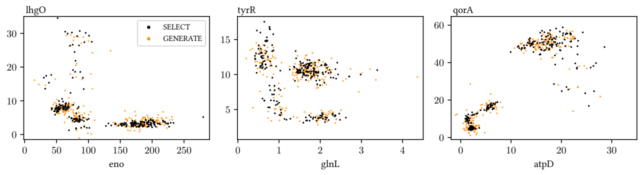
(c) Add IPTG but not Arabinose



(d) 6 genes from (c).

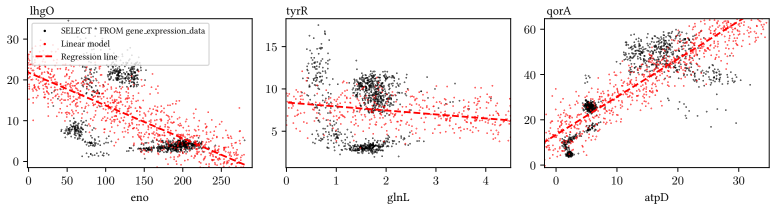
```
SELECT * FROM gene_expression_data
WHERE Ara. = "added" AND IPTG = "added"

GENERATE UNDER gene_expression_model
  GIVEN Ara. = "added" AND IPTG = "added"
LIMIT 1000
```

(e) Add both IPTG and Arabinose

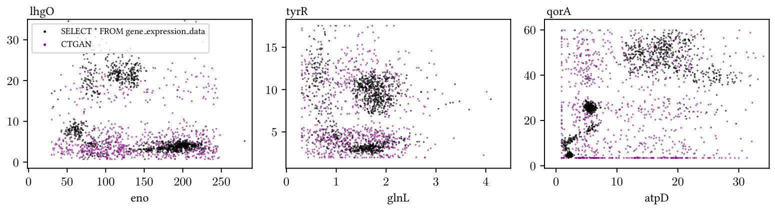


(f) 6 genes from (e).

Fig. 13. Case study: Conditional synthetic data generation for a virtual wet lab.



(a) Real and generated data from bivariate linear models (compare to Fig. 13b).



(b) Comparison of real and generated data from CTGAN (compare to Fig. 13b).

Fig. 14. Linear models and conditional generative adversarial networks (CTGAN [87]) produce less accurate synthetic virtual wet lab data as compared to the synthetic data from GenSQL shown in Fig. 13b. In (b), the default model and inference parameters in the open source implementation of CTGAN is used.

apply a **WHERE** filter that removes BMI values outside of the 5th and the 95th percentile, excluding one-dimensional extreme values (Fig. 12a). We then compute the conditional probabilities of the BMI values in each row for the remaining data and return anomalies. Fig. 12c shows the posterior predictive over the ensemble of models (line) and for each individual model (dots) for a BMI above 30 given exercise and health status.

Conditional synthetic data generation for virtual wet lab. Figure 13 shows synthetic gene expression data generated using GenSQL, given a dataset from an experiment testing genetic circuits [56] in *Escherichia coli*. The synthetic data aligns with the overall population characteristics (Figs. 13a and 13b) and accurately reflects the outcomes of actual experimental interventions (Figs. 13c to 13f). In synthetic biology, the prospect of implementing genetic circuits has fundamental implications for medical device engineering [53], bio-sensing [81] and environmental biotechnology [88]. These circuits require input which is typically provided by adding inducer substances to the culture mediums where the organisms are grown. Our figures show the effect of adding two such inducer substances, Arabinose and IPTG on 6 different host genes.

Producing standard RNA sequencing data can be costly [48], especially for new, engineered organisms that are not mass-produced. When it is produced, RNA sequencing will yield measurements for gene expressions for thousands of annotated host genes [9]. These genes are highly interrelated, and knowledge of the relations is only partially available [44]. Thus, the application of generative models to these data presents a challenging high-dimensional modeling problem, further compounded by the inherent non-linearity in the data, as illustrated in Figs. 14a and 14b.

The most popular approach to modeling gene expression data is linear regression [22], as models are easy to interpret and readily available in data analysis libraries. For non-numerical data, linear regression requires analysis-specific re-coding of discrete values. That aside, the low capacity of the model means that it fails to faithfully reproduce in the actual wet lab data, as shown in Fig. 14a. A more complex approach to generating synthetic expression data is using conditional generative adversarial networks (CTGAN) [87]. CTGANs are an appropriate baseline because they are domain-general and effective at modeling multivariate, heterogeneous data. However, GANs are hard to interpret and as RNA sequencing data acquisition is so costly, the number of available training examples (943) renders it unsuitable for CTGANs. Fig. 14b depicts this model class failing to accurately model the gene expression data.

6 RELATED WORK

Probabilistic databases. Probabilistic databases systems [78, 80] develop efficient algorithms for inference queries on discrete distributions over databases, often based on variants of weighted model counting, for which hardness complexity results were shown and algorithms were developed for tractable cases and efficient approximations. Cambronero et al. [13] integrate probabilities into a relational database system to support imputation, while Hilprecht et al. [39] use probabilistic circuits to improve query performance. Jampani et al. [42] use probabilistic databases to support random data generation and simulation. Cai et al. [12] provides Gibbs sampling support in the space of database tables to a SQL-like language, enabling bayesian machine learning workload such as linear regression or latent Dirichlet allocation. These languages are typically extensions to SQL or relational algebra but with limited support for probabilistic models, which they tradeoff for performance. Schaechtle et al. [76] presents a preliminary design for extending SQL to support probabilistic models of tabular data. Our work differs in that it presents (1) a formalization of the system; (2) a denotational semantics; (3) soundness guarantees for the system; (4) a unified interface that probabilistic models implement; (5) a lowering transform and target lowering language; (6) an extensive performance evaluation; and (7) two new case studies on real-world data.

Semantics of probabilistic databases. Bárány et al. [3] and Grohe et al. [34] give a semantic account to probabilistic databases by giving a probabilistic semantics and guarantees to an extension of Datalog. Dash and Staton [21] give a monadic account and denotational semantics for measurable queries in probabilistic databases. Their semantics of SQL-like expressions inspired the semantics of our table expressions. Grohe and Lindner [35] established a formal framework for reasoning about infinite probabilistic databases. Benzaken and Contejean [5] formalized the semantics of SQL in Coq while Borya [11] formalized relational algebra and a SQL-like syntax using a model checker.

Probabilistic program synthesis. GenSQL has been designed with the possibility to leverage powerful probabilistic program synthesis techniques based on Bayesian [1, 50, 69] or non-Bayesian [17, 30, 36, 41, 57] probabilistic model discovery. The AMI provides a unifying approach to expressing powerful Bayesian inference workflows in these probabilistic programs using a high-level SQL-like language. Extending the interface to handle synthesized models of time series [70, 72] and/or relational data [71] is a promising avenue for future work.

Probabilistic programming systems. While we used a Clojure version [16] of CrossCat [50] in our experiments, our system supports any probabilistic program that satisfies the rowModel interface. We can thus reuse models written in the variety of PPLs developed in the literature, such as models written in languages supporting approximate inference [8, 14, 20, 26, 54, 75, 84] and exact inference [27, 40, 73, 89]. Our model interface is inspired by the SPPL interface [73] and the CGPM interface [68]. Gordon et al. [33] propose a probabilistic programming system using a functional syntax similar to the stochastic lambda calculus, specialized to inference over relational databases, implemented on top on Infer.net. It can perform inference tasks such as linear regression and querying for missing values which enable data imputation, classification, or clustering. Borgström et al. [10] present a probabilistic DSL and semantics for regression formulas in the style of the formula DSL in R. Domain-specific PPLs for tabular data have also been designed to solve tasks such as data cleaning [46, 63].

BayesDB. Although BayesDB [52] was motivated by similar goals as GenSQL, GenSQL introduces novel semantics concepts and soundness theorems that BayesDB did not. GenSQL also improves upon BayesDB in terms of expressiveness and performance, as shown in Section 5. For example, GenSQL queries can be nested and interleaved with SQL, and also combine results from multiple models. GenSQL also provides an exact inference engine for a broad class of sum-product probabilistic programs [73]. BayesDB, on the other hand, has interesting features that GenSQL does not yet support such as iterating over model and columns (e.g. to find pairs of columns with the highest mutual information) [67] and similarity search between rows [65]. BayesDB also has a "meta-modeling" DSL [66] for composing probabilistic programs from various sources.

Automated Machine Learning. Several systems [6, 23, 43, 45, 58, 79] have been developed to automate the use of discriminative machine learning methods for analyzing tabular data. Unlike GenSQL, they do not support the use of generative probabilistic programs for tabular data satisfying a unified interface (for sampling, conditioning, and evaluating probabilities or densities) which enables a single model to be reused across many different tasks.

7 CONTRIBUTIONS

GenSQL specializes probabilistic programming languages to applications with tabular data. It is differentiated from general purpose PPLs in three main ways:

- **Through the AMI, GenSQL enables multi-language workflows.** Users from different domains and with different expertise should be able to use probabilistic models for their queries without having to learn all the details of the PPL in which the model is written. The AMI enables this separation of concerns by providing a well-specified interface. It enables the integrating probabilistic models of tabular data in different languages, as it can be implemented in either a general-purpose or domain-specific PPL (Appendix C). There is no standard way to jointly query models in different PPLs or use the result of a query in one language against a model in another language. As different PPLs focus on different workloads, users of GenSQL can work with several models written in different PPLs. GenSQL thus provides a natural multi-language workflow, and our experiments already use multiple backends (Gen.clj, SPPL, and ClojureCat).

- **GenSQL enables declarative querying.** No current PPL offers a simple declarative syntax for evaluating complex queries (e.g., containing elaborate joins and nested selects) interleaving calls on probabilistic models and database tables. A number of PPLs provide declarative syntax for specifying and conditioning models, but the user must decide which operations on what conditional distributions to evaluate and then manually combine the results of these operations. GenSQL relieves the users of such concerns, reducing the chances of programming errors.
- **GenSQL enables reusable performance optimizations.** Widely used database management systems (DBMS) have been optimized by many engineer-hours of effort over several decades. These optimizations are highly reusable because they are independent of the application domain and specific languages that the DBMS interfaces with. GenSQL enables analogous optimizations for workloads that interleave ordinary database queries with probabilistic inference and generative modeling. GenSQL's optimizations can carry over to many domains and workflows, avoiding the need for project-specific performance optimizations involving probabilistic models of tabular data.

We see two interesting avenues for GenSQL to impact database applications and design.

Integration of GenSQL with database management systems (DBMS). First, GenSQL could serve as a query language, allowing users to query generative models of tabular data directly from the DBMS. One use case of rapidly increasing practical importance is querying synthetic data, generated on the fly to meet user-specified privacy-utility trade-offs, instead of querying real data that cannot be shared due to privacy constraints. Other potential applications for synthetic data include testing, performance tuning, and sensitivity analysis of end-to-end data analysis workflows. In all these settings, GenSQL implementations could also draw on performance engineering innovations from DBMS engines, optimized further using the generative models themselves (e.g., to reduce variance for stratified sampling approximations to SQL aggregates [2]).

Modularized development of queries and models. GenSQL introduces abstractions that isolate query developers and query users from model developers. This separation of concerns is analogous to the physical data independence property achieved by relational databases [19]. Most database users do not need to know the details of how data is stored and indexed to be able to query it efficiently, but some experts do understand how to tune indices to ensure that databases meet the necessary performance constraints. Most GenSQL users need not be experts on the details of the algorithms, modeling assumptions, and software pipelines that produced the underlying generative models. Expert statisticians and generative modelers can still ensure the models are of sufficient quality and tune trade-offs between performance, maintenance costs, and accuracy, improving models without invalidating user workflows. With GenSQL, both typical users and experts can more easily and interactively query generative models to test their validity, both qualitatively and quantitatively. This division of responsibility between users, generative modelers, and probabilistic programming system developers could potentially help our society more safely and productively broaden the deployment of generative models for tabular data.

ACKNOWLEDGMENTS

MIT contributors would like to acknowledge support from DARPA, under the Machine Common Sense (MCS) program (Award ID: 030523-00001) Synergistic Discovery and Design (SD2) program (Contract No. FA8750-17-C-0239), and Compositionally Organized Learning To Reason About Novel Experiences (COLTRANE) grant (Contract No. 140D0422C0045); and unrestricted gifts from Google, an anonymous donor, and the Siegel Family Foundation. The authors wish to thank the anonymous referees for their valuable comments and suggestions. We have also benefited greatly from conversations with and feedback from Timothy J. O'Donnell, João Loula and Pierre Glaser.

REFERENCES

[1] Ryan P. Adams, Hanna Wallach, and Zoubin Ghahramani. 2010. Learning the Structure of Deep Sparse Graphical Models. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*. PMLR, Norfolk, MA, USA, 1–8.

[2] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Stoica Ion. 2013. BlinkDB: Queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, New York, NY, USA, 29–42. https://doi.org/10.1145/2465351.2465355

[3] Vince Bárány, Balder Ten Cate, Benny Kimelfeld, Dan Olteanu, and Zografoula Vagena. 2017. Declarative Probabilistic Programming with Datalog. *ACM Transactions on Database Systems* 42, 4 (2017), 1–35. https://doi.org/10.1145/3132700

[4] Luc Bauwens, Michel Lubrano, and Jean-Francois Richard. 2000. *Bayesian Inference in Dynamic Econometric Models*. Oxford University Press, Oxford, UK.

[5] Véronique Benzaken and Evelyne Contejean. 2019. A Coq Mechanised Formal Semantics For Realistic SQL Queries: Formally Reconciling SQL and Bag Relational Algebra. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*. ACM, New York, NY, USA, 249–261. https://doi.org/10.1145/3293880.3294107

[6] James Bergstra, Brent Komer, Chris Eliasmith, Dan Yamins, and David D Cox. 2015. Hyperopt: A Python Library for Model Selection And Hyperparameter Optimization. *Computational Science & Discovery* 8, 1 (2015), 014008. https://doi.org/10.1088/1749-4699/8/1/014008

[7] Patrick Billingsley. 1995. *Probability and Measure* (3rd ed.). John Wiley & Sons, New York.

[8] Eli Bingham, Jonathan P Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D Goodman. 2019. Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research* 20, 1 (2019), 973–978.

[9] Frederick R. Blattner et al. 1997. The Complete Genome Sequence of Escherichia Coli K-12. *Science* 277, 5331 (1997), 1453–1462. https://doi.org/10.1126/science.277.5331.1453

[10] Johannes Borgström, Andrew D. Gordon, Long Ouyang, Claudio Russo, Adam Scibior, and Marcin Szymczak. 2016. Fabular: Regression Formulas as Probabilistic Programming. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM, New York, NY, USA, 271–283. https://doi.org/10.1145/2837614.2837653

[11] Joachim Borya. 2023. *Formalisation of Relational Algebra and a SQL-like Language with the RISCAL Model Checker*. Technical Report 23-06. Research Institute for Symbolic Computation (RISC), Johannes Kepler University Linz. https://doi.org/10.35011/risc.23-06

[12] Zhuhua Cai, Zografoula Vagena, Luis Perez, Subramanian Arumugam, Peter J. Haas, and Christopher Jermaine. 2013. Simulation of Database-Valued Markov Chains Using SimSQL. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, New York, NY, USA, 637–648. https://doi.org/10.1145/2463676.2465283

[13] José Cambronero, John K. Feser, Micah J. Smith, and Samuel Madden. 2017. Query Optimization for Dynamic Imputation. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1310–1321. https://doi.org/10.14778/3137628.3137641

[14] Bob Carpenter, Andrew Gelman, Matthew D. Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus A. Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. 2017. Stan: A Probabilistic Programming Language. *Journal of Statistical Software* 76, 1 (2017), 1–32. https://doi.org/10.18637/jss.v076.i01

[15] CDC. 2022. BMI Percentile Calculator: Body Mass Indexes in the United States. https://dqydj.com/bmi-percentile-calculator-united-states

[16] Nicholas G. Charchut. 2020. *Implementation of a Cross-Platform Automated Bayesian Data Modeling System*. Master's thesis. Massachusetts Institute of Technology, Cambridge, MA.

[17] Sarah Chasins and Phitchaya M. Phothilimthana. 2017. Data-driven synthesis of full probabilistic programs. In *Proceedings of the 29th International Conference on Computer Aided Verification*. Springer, Cham, 279–304. https://doi.org/10.1007/978-3-319-63387-9_14

[18] Nicolas Chopin, Omiros Papaspiliopoulos, et al. 2020. *An Introduction to Sequential Monte Carlo*. Springer, Cham. https://doi.org/10.1007/978-3-030-47845-2

[19] Edgar F. Codd. 1970. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM* 13, 6 (1970), 377–387. https://doi.org/10.1145/362384.362685

[20] Marco F. Cusumano-Towner, Feras A. Saad, Alexander K. Lew, and Vikash K. Mansinghka. 2019. Gen: A general-purpose probabilistic programming system with programmable inference. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, New York, NY, USA, 221–236. https://doi.org/10.1145/3314221.3314642

[21] Swaraj Dash and Sam Staton. 2021. Monads for Measurable Queries in Probabilistic Databases. arXiv:2112.14048

[22] Sebastian Eck and Wolfgang Stephan. 2008. Determining the Relationship of Gene Expression and Global mRNA Stability in Drosophila Melanogaster And Escherichia Coli Using Linear Models. *Gene* 424, 1-2 (2008), 102–107. https://doi.org/10.1016/j.gene.2008.07.033

[23] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. 2020. AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data. arXiv:2003.06505

[24] D. H. Fremlin. 2001. *Measure Theory*. Vol. 2. Torres Fremlin, Colchester, England.

[25] Jonah Gabry, Daniel Simpson, Aki Vehtari, Michael Betancourt, and Andrew Gelman. 2019. Visualization in Bayesian Workflow. *Journal of the Royal Statistical Society Series A: Statistics in Society* 182, 2 (2019), 389–402. https://doi.org/10.1111/rssa.12378

[26] Hong Ge, Kai Xu, and Zoubin Ghahramani. 2018. Turing: A Language for Flexible Probabilistic Inference. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics*. PMLR, Norfolk, MA, USA, 1682–1690.

[27] Timon Gehr, Sasa Misailovic, and Martin Vechev. 2016. PSI: Exact Symbolic Inference for Probabilistic Programs. In *Proceedings of the 28th International Conference on Computer Aided Verification*. Springer, Cham, 62–83. https://doi.org/10.1007/978-3-319-41528-4_4

[28] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. 2014. *Bayesian Data Analysis* (3rd ed.). CRC Press, Boca Raton. https://doi.org/10.1201/9780429258411

[29] Andrew Gelman, Aki Vehtari, Daniel Simpson, Charles C. Margossian, Bob Carpenter, Yuling Yao, Lauren Kennedy, Jonah Gabry, Paul-Christian Bürkner, and Martin Modrák. 2020. Bayesian Workflow. arXiv:2011.01808

[30] Robert Gens and Pedro Domingos. 2013. Learning the Structure of Sum-Product Networks. In *Proceedings of the 30th International Conference on Machine Learning*. PMLR, Norfolk, MA, USA, 873–880.

[31] Alan Genz and Frank Bretz. 2009. *Computation of Multivariate Normal and t Probabilities*. Lecture Notes in Statistics, Vol. 195. Springer, Cham. https://doi.org/10.1007/978-3-642-01689-9

[32] Michele Giry. 2006. A Categorical Approach To Probability Theory. In *Categorical Aspects of Topology and Analysis: Proceedings of an International Conference*. Springer, Cham, 68–85. https://doi.org/10.1007/BFb0092872

[33] Andrew D. Gordon, Thore Graepel, Nicolas Rolland, Claudio Russo, Johannes Borgström, and John Guiver. 2014. Tabular: A Schema-Driven Probabilistic Programming Language. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM, New York, NY, USA, 321–334. https://doi.org/10.1145/2535838.2535850

[34] Martin Grohe, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Peter Lindner. 2022. Generative Datalog with Continuous Distributions. *J. ACM* 69, 6 (2022), 1–52. https://doi.org/10.1145/3559102

[35] Martin Grohe and Peter Lindner. 2022. Infinite Probabilistic Databases. *Logical Methods in Computer Science* 18, 1, Article 34 (2022), 43 pages. https://doi.org/10.46298/LMCS-18(1:34)2022

[36] Roger Grosse, Ruslan Salakhutdinov, William Freeman, and Joshua B. Tenenbaum. 2012. Exploiting compositionality to explore a large space of model structures. In *Proceedings of the 28th Annual Conference on Uncertainty in Artificial Intelligence*. AUAI Press, Puyallup, WA, USA, 306–315.

[37] Allen Hatcher. 2002. *Algebraic Topology*. Cambridge University Press, Cambridge, UK.

[38] Miguel A. Hernán and James M. Robins. 2006. Estimating Causal Effects from Epidemiological Data. *Journal of Epidemiology & Community Health* 60, 7 (2006), 578–586. https://doi.org/10.1136/jech.2004.029496

[39] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2020. DeepDB: Learn from Data, Not from Queries! *Proc. VLDB Endow.* 13, 7 (2020), 992–1005. https://doi.org/10.14778/3384345.3384349

[40] Steven Holtzen, Guy Van den Broeck, and Todd Millstein. 2020. Scaling Exact Inference for Discrete Probabilistic Programs. *Proc. ACM Program. Lang.* 4, OOPSLA, Article 140 (2020), 31 pages. https://doi.org/10.1145/3133904

[41] Irvin Hwang, Andreas Stuhlmüller, and Noah D Goodman. 2011. Inducing Probabilistic Programs by Bayesian Program Merging. arXiv:1110.5667

[42] Ravi Jampani, Fei Xu, Mingxi Wu, Luis Perez, Chris Jermaine, and Peter J. Haas. 2011. The Monte Carlo Database System: Stochastic Analysis Close to the Data. *ACM Transactions on Database Systems* 36, 3 (2011), 1–41. https://doi.org/10.1145/2000824.2000828

[43] Haifeng Jin, Qingquan Song, and Xia Hu. 2019. Auto-Keras: An Efficient Neural Architecture Search System. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, New York, NY, USA, 1946–1956. https://doi.org/10.1145/3292500.3330648

[44] Ingrid M Keseler, Socorro Gama-Castro, Amanda Mackie, Peter E Midford, Alan J Wolfe, Julio Collado-Vides, Ian T Paulsen, and Peter D Karp. 2021. The EcoCyc Database in 2021. *Frontiers in Microbiology* 12 (2021), 711077. https://doi.org/10.3389/fmicb.2021.711077

[45] Erin LeDell and Sebastien Poirier. 2020. H2O AutoML: Scalable Automatic Machine Learning. In *Proceedings of the 7th ICML Workshop on AutoML*. AutoML-Conf, 16 pages.

[46] Alexander Lew, Monica Agrawal, David Sontag, and Vikash Mansinghka. 2021. PClean: Bayesian Data Cleaning at Scale With Domain-Specific Probabilistic Programming. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*. PMLR, Norfolk, MA, USA, 1927–1935.

[47] Alexander K Lew, Matin Ghavamizadeh, Martin C Rinard, and Vikash K Mansinghka. 2023. Probabilistic Programming with Stochastic Probabilities. *Proceedings of the ACM on Programming Languages* 7, PLDI (2023), 1708–1732. https:

//doi.org/10.1145/3591290
[48] Brian K. Lohman, Jesse N. Weber, and Daniel I. Bolnick. 2016. Evaluation Of Tagseq, A Reliable Low-Cost Alternative for RNA Seq. *Molecular Ecology Resources* 16, 6 (2016), 1315–1321. https://doi.org/10.1111/1755-0998.12529
[49] David JC MacKay et al. 1998. Introduction to Gaussian Processes. *NATO ASI series F computer and systems sciences* 168 (1998), 133–166.
[50] Vikash Mansinghka, Patrick Shafto, Eric Jonas, Cap Petschulat, Max Gasner, and Joshua B. Tenenbaum. 2016. CrossCat: A Fully Bayesian Nonparametric Method for Analyzing Heterogeneous, High Dimensional Data. *Journal of Machine Learning Research* 17, 138 (2016), 1–49.
[51] Vikash K. Mansinghka, Ulrich Schaechtle, Shivam Handa, Alexey Radul, Yutian Chen, and Martin Rinard. 2018. Probabilistic programming with programmable inference. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, New York, NY, USA, 603–616. https://doi.org/10.1145/3192366.3192409
[52] Vikash K. Mansinghka, Richard Tibbetts, Jay Baxter, Pat Shafto, and Baxter Eaves. 2015. BayesDB: A Probabilistic Programming System for Querying the Probable Implications of Data. arXiv:1512.05006
[53] Maysam Mansouri and Martin Fussenegger. 2022. Therapeutic Cell Engineering: Designing Programmable Synthetic Genetic Circuits in Mammalian Cells. *Protein & Cell* 13, 7 (2022), 476–489. https://doi.org/10.1007/s13238-021-00876-1
[54] Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L. Ong, and Sontag Kolobov. 2005. BLOG: Probabilistic models with unknown objects. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, San Francisco, CA, USA, 1352–1359.
[55] Kevin P. Murphy. 2022. *Probabilistic Machine Learning: An Introduction*. MIT Press, Cambridge, MA.
[56] Alec A. K. Nielsen, Bryan S. Der, Jonghyeon Shin, Prashant Vaidyanathan, Vanya Paralanov, Elizabeth A. Strychalski, David Ross, Douglas Densmore, and Christopher A. Voigt. 2016. Genetic circuit design automation. *Science* 352, 6281 (2016), aac7341. https://doi.org/10.1126/science.aac7341
[57] Aditya V. Nori, Sherjil Ozair, Sriram K. Rajamani, and Deepak Vijaykeerthy. 2015. Efficient synthesis of probabilistic programs. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, New York, NY, USA, 208–217. https://doi.org/10.1145/2737924.2737982
[58] Randal S. Olson, Ryan J. Urbanowicz, Peter C. Andrews, Nicole A. Lavender, La Creis Kidd, and Jason H. Moore. 2016. Automating Biomedical Data Science Through Tree-Based Pipeline Optimization. In *Applications of Evolutionary Computation*. Springer, Cham, 123–137. https://doi.org/10.1007/978-3-319-31204-0_9
[59] Judea Pearl. 1988. *Probabilistic Reasoning In Intelligent Systems: Networks Of Plausible Inference*. Morgan Kaufmann, San Francisco, CA, USA.
[60] Fabian Pedregosa et al. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning research* 12 (2011), 2825–2830.
[61] MIT Probabilistic Computing Project. 2024. Gen.clj. https://github.com/probcomp/Gen.clj
[62] Tom Rainforth. 2018. Nesting Probabilistic Programs. In *Proceedings of the 34th Annual Conference on Uncertainty in Artificial Intelligence*. AUAI Press, Puyallup, WA, USA, 10 pages. arXiv:1803.06328
[63] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1190–1201. https://doi.org/10.14778/3137628.3137631
[64] Christian P. Robert and George Casella. 2004. *Monte Carlo Statistical Methods* (2 ed.). Springer, Cham. https://doi.org/10.1007/978-1-4757-4145-2
[65] Feras Saad, Leonardo Casarsa, and Vikash Mansinghka. 2017. Probabilistic Search for Structured Data via Probabilistic Programming and Nonparametric Bayes. arXiv:1704.01087
[66] Feras Saad and Vikash Mansinghka. 2016. Probabilistic Data Analysis with Probabilistic Programming. arXiv:1608.05347
[67] Feras Saad and Vikash Mansinghka. 2017. Detecting Dependencies in Sparse, Multivariate Databases using Probabilistic Programming and Non-Parametric Bayes. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. PMLR, Norfolk, MA, USA, 632–641.
[68] Feras Saad and Vikash K. Mansinghka. 2016. A Probabilistic Programming Approach to Probabilistic Data Analysis. In *Advances in Neural Information Processing Systems*, Vol. 29. Curran Associates, Inc., Red Hook, NY, USA, 2011–2019.
[69] Feras A. Saad, Marco F. Cusumano-Towner, Ulrich Schaechtle, Martin C. Rinard, and Vikash K. Mansinghka. 2019. Bayesian Synthesis of Probabilistic Programs for Automatic Data Modeling. *Proceedings of the ACM on Programming Languages* 3, POPL, Article 37 (2019), 29 pages. https://doi.org/10.1145/3290350
[70] Feras A. Saad and Vikash K. Mansinghka. 2018. Temporally-Reweighted Chinese Restaurant Process Mixtures for Clustering, Imputing, and Forecasting Multivariate Time Series. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics*. PMLR, Norfolk, MA, USA, 755–764.
[71] Feras A. Saad and Vikash K. Mansinghka. 2021. Hierarchical Infinite Relational Model. In *Proceedings of the 37th Conference on Uncertainty in Artificial Intelligence*. PMLR, Norfolk, MA, USA, 1067–1077.

[72] Feras A. Saad, Brian J. Patton, Matthew D. Hoffmann, Rif A. Saurous, and Vikash K. Mansinghka. 2023. Sequential Monte Carlo Learning for Time Series Structure Discovery. In *Proceedings of the 40th International Conference on Machine Learning*. PMLR, Norfolk, MA, USA, Article 1226, 17 pages.

[73] Feras A. Saad, Martin C. Rinard, and Vikash K. Mansinghka. 2021. SPPL: Probabilistic Programming with Fast Exact Symbolic Inference. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. ACM, New York, NY, USA, 804–819. https://doi.org/10.1145/3453483.3454078

[74] Feras A. K. Saad. 2022. *Scalable Structure Learning, Inference, and Analysis with Probabilistic Programs*. Ph. D. Dissertation. Massachusetts Institute of Technology.

[75] John Salvatier, Thomas V. Wiecki, and Christopher Fonnesbeck. 2016. Probabilistic Programming in Python using PyMC3. *PeerJ Computer Science* 2 (2016), e55. https://doi.org/10.7717/peerj-cs.55

[76] Ulrich Schaechtle, Cameron Freer, Zane Shelby, Feras Saad, and Vikash Mansinghka. 2022. Bayesian AutoML for Databases via the InferenceQL Probabilistic Programming System. In *Proceedings of the 1st Conference on Automated Machine Learning (Late-Breaking Workshop)*. AutoML-Conf, 8 pages.

[77] Chung-chieh Shan and Norman Ramsey. 2017. Exact Bayesian Inference by Symbolic Disintegration. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*. ACM, New York, NY, USA, 130–144. https://doi.org/10.1145/3009837.3009852

[78] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. 2011. *Probabilistic Databases*. Springer, Cham. https://doi.org/10.1007/978-3-031-01879-4

[79] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: Combined Selection and Hyperparameter Optimization Of Classification Algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, New York, NY, USA, 847–855. https://doi.org/10.1145/2487575.2487629

[80] Guy Van den Broeck and Dan Suciu. 2017. Query Processing on Probabilistic Data: A Survey. *Foundations and Trends in Databases* 7, 3-4 (2017), 197–341. https://doi.org/10.1561/1900000052

[81] Baojun Wang, Mauricio Barahona, and Martin Buck. 2013. A Modular Cell-Based Biosensor Using Engineered Genetic Logic Circuits to Detect and Integrate Multiple Environmental Signals. *Biosensors and Bioelectronics* 40, 1 (2013), 368–376. https://doi.org/10.1016/j.bios.2012.08.011

[82] WHO. 2023. World Health Organization: Obesity. https://www.who.int/health-topics/obesity

[83] Darren J. Wilkinson. 2018. *Stochastic Modelling for Systems Biology*. CRC Press, Boca Raton, FL, USA.

[84] Frank Wood, Jan Willem Meent, and Vikash Mansinghka. 2014. A New Approach To Probabilistic Programming Inference. In *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics*. PMLR, Norfolk, MA, USA, 1024–1032.

[85] Yi Wu, Siddharth Srivastava, Nicholas Hay, Simon Du, and Stuart Russell. 2018. Discrete-Continuous Mixtures in Probabilistic Programming: Generalized Semantics and Inference Algorithms. In *Proceedings of the 35th International Conference on Machine Learning*. PMLR, Norfolk, MA, USA, 5343–5352.

[86] xCures. 2019. BEAT19 Behavior, Environment And Treatments for COVID-19. https://classic.clinicaltrials.gov/ct2/show/results/NCT04321811?view=results

[87] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. 2019. Modeling Tabular Data using Conditional GAN. In *Advances in Neural Information Processing Systems*, Vol. 32. Curran Associates, Inc., Red Hook, NY, USA, 11 pages.

[88] Yubin Xue, Pei Du, Amal Amin Ibrahim Shendi, and Bo Yu. 2022. Mercury Bioremediation in Aquatic Environment by Genetically Modified Bacteria with Self-Controlled Biosecurity Circuit. *Journal of Cleaner Production* 337 (2022), 130524. https://doi.org/10.1016/j.jclepro.2022.130524

[89] Fabian Zaiser, Andrzej Murawski, and Chih-Hao Luke Ong. 2023. Exact Bayesian Inference on Discrete Models via Probability Generating Functions: A Probabilistic Programming Approach. In *Advances in Neural Information Processing Systems*, Vol. 36. Curran Associates, Inc., Red Hook, NY, USA, 2427–2462.

A FURTHER EXPERIMENTS AND COMPARISONS

A.1 Code comparison with Scikit-learn

Figure 15 shows the implementation of a single-line GenSQL query

SELECT class FROM GENERATE UNDER model GIVEN petal width = 0.1 LIMIT 100

in Scikit-learn [60] on the Iris data from the UCI ML repository. Even though the implementation contains data loading and preprocessing, model building, and a few comments, the model querying alone is more than 50 lines long and clearly error prone. GenSQL offers a significant advantage in simplicity over such baselines.

The reason is that Scikit-learn is a Python library that focuses on predictive modeling, not a platform for interactive querying of probabilistic models. We now detail some aspects of the implementation in Scikit-learn. Heterogeneously typed mixture models are closely related to the AMI specification we use for many experiments in the model. The Scikit-learn implementations does not natively support discrete columns. Instead, Scikit-learn advises to apply column transformations like one-hot encoding. This makes sense for discriminative machine learning pipelines but less so for multivariate generative models where users want to refer to columns and their discrete values repeatedly and in two directions (what goes into the model for conditioning and what comes out). More importantly, conditioning models on partial information is not supported in Scikit-learn. The predict_proba method for mixture models in Scikit-learn returns updated weights for latent components which are at the core of conditioning such a model. Yet, the predict_proba method crashes with partial input. Users have to implement such methods manually before they can refer to the sample method and post-process its output for the readability of discrete columns.

A.2 Comparison against a baseline using approximate inference

In this section, we present some more simple experiments comparing GenSQL using an exact backend (SPPL) with a Gen.clj baseline using approximate inference. The results obtained are expected and confirm the fact that for simple but non-trivial models, exact inference can be both faster and more accurate than approximate inference. This is especially true for rare events for which getting accurate estimates can be crucial in domains such as risk assessment, fraud detection, and rare disease diagnosis. The fact that in GenSQL one can easily change the backend models is therefore a significant advantage, where one can use approximate models when the exact model is too slow, or switch to an exact model when the approximate model is not accurate enough. We compare the normalized standard deviation of the estimator and the runtime of the queries. We use the same probabilistic model for both backends, and we compare the results on a PROBABILITY OF c^0 query on a possibly conditioned model.

Fig. 16 presents a runtime and normalized standard deviation of the estimator comparison. We compare an exact and an approximate backend using importance sampling on a query

PROBABILITY OF c^0 UNDER m GIVEN c_i^0

with varying c_i^0. The same probabilistic model is used, using exact inference for GenSQL and importance sampling (blue: 5000, red: 10000 importance samples) in Gen.clj. GenSQL is faster by orders of magnitude and more accurate than the baseline Gen.clj model (runtime: in the presence of at least one condition).

A.3 Gen.clj code for emission functions

Fig. 17 shows the Gen.clj code for a simple population model and a harmonized model with emission functions that uses age categories as opposed to integers for age. Such emissions functions are useful for harmonizing models with different data sources. For instance, one model can be trained

```python
import numpy as np
import pandas as pd
from scipy.stats import norm
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.mixture import GaussianMixture
from sklearn.pipeline import make_pipeline
from ucimlrepo import fetch_ucirepo

###### Fetch Iris data from UCI ML repo #####
iris = fetch_ucirepo(id=53)
data = iris.data["original"]

###### Model building #####
# Define the preprocessing for the categorical features.
categorical_features = ["class"]
categorical_transformer = OneHotEncoder(handle_unknown="ignore")

# Create the ColumnTransformer to apply the transformations to the correct
# columns.
preprocessor = ColumnTransformer(
    transformers=[("categorical-columns", categorical_transformer, categorical_features)],
    remainder="passthrough")

# Use SKlearn to create a mixture model with heterogenous types;
# Gaussians are a bad choice of primitive here but this is
# supposed to be an illustrative example... SKlearn does not actually
# support heterogenously typed mixtures.
pipeline = make_pipeline(
    preprocessor, GaussianMixture(n_components=3, covariance_type="diag")
    ).fit(data)

###### Model querying: GENERATE class UNDER model GIVEN petal width = 0.1 #####
def get_normal_params(component_idx, pipeline, condition_name):
    # Get index of the column in the sklearn internals. The first columns
    # are the one-hot encoded columns, then we add the original index.
    col_index = data["class"].unique().shape[0] + data.columns.get_loc(condition_name)
    # 2. Read out mu.
    m = pipeline.named_steps["gaussianmixture"].means_[component_idx, col_index]
    # 3. Read out sigma.
    sigma = np.sqrt(
        pipeline.named_steps["gaussianmixture"].covariances_[component_idx, col_index])
    return {"mu": m, "sigma": sigma}

def get_comp_score(val, params):
    return norm.pdf(val, params["mu"], params["sigma"])
```

Fig. 15. Scikit-learn implementation of a simple GenSQL query.

on data with age as an integer, while another model can be trained on data with age categories. These two values are not directly comparable, but emission functions can be used to harmonize them, and therefore allow for querying across models trained on different data sources. While the given example is simple, they are essential for real-world applications where data sources are often heterogeneous.

```
def condition_model(pipeline, condition_name, condition_value):
    unnormalized_updated_weights = pipeline.named_steps["gaussianmixture"].weights_
        * np.asarray([
            get_comp_score(
                condition_value, get_normal_params(0, pipeline, condition_name)),
            get_comp_score(
                condition_value, get_normal_params(1, pipeline, condition_name)),
            get_comp_score(
                condition_value, get_normal_params(2, pipeline, condition_name)),
        ])

    updated_weights = unnormalized_updated_weights / sum(unnormalized_updated_weights)
    pipeline.named_steps["gaussianmixture"].weights_ = updated_weights
    return pipeline

# Sample 100 times from conditional model.
conditioned_pipeline = condition_model(
    pipeline, condition_name="petal width", condition_value=0.1)
synethetic_data_with_unreadable_categoricals, _ = conditioned_pipeline.named_steps[
    "gaussianmixture"].sample(100)

# Back transform one-hot encoded class column to readiable iris names.
print(preprocessor
    .named_transformers_["categorical-columns"]
    .inverse_transform(X=synethetic_data_with_unreadable_categoricals[:, -3:]))
```

Fig. 15. Scikit-learn implementation of a simple GenSQL query (continued).

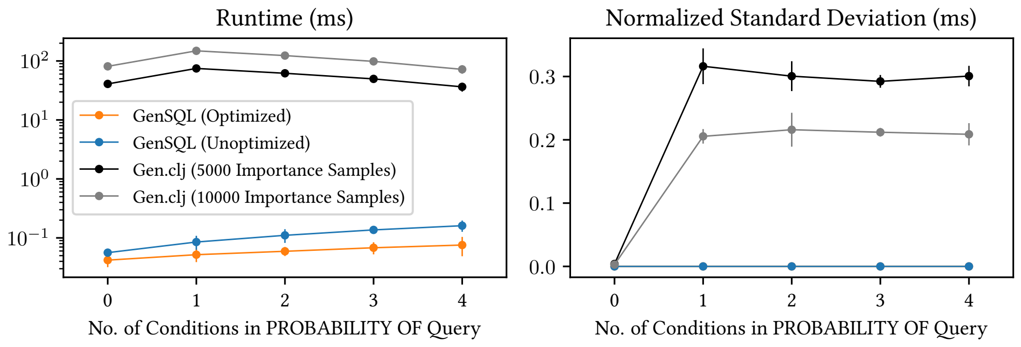


Fig. 16. Comparing the effect on runtime (left) and on standard deviation of the estimated probability (right) between GenSQL and Gen.clj [20] on a simple probability query, varying the number of conditions.

B DETAILS OF THE OPTIMIZATIONS

A dominating computational factor in GenSQL queries for exact models is often model conditioning, analogously to the role of join computations in SQL. For these models, this usually amounts to a program transformation [73], or e.g., a costly matrix inversion [49]. Likewise, for both exact and approximate models, a typical GenSQL workload will involve repeated likelihood evaluations and querying a conditioned rowModel repeatedly over the rows of a data table. We can optimize this

```
;; This function extends Gen's GFI to the AMI using 1000 importance samples for each query.
(gen->iql 1000)

;; Create a simple population model for exposition
(def population-model
  (with-meta ;; users have to specify all variables in the Gen model that will be queried.
    (gen
      []
      (let [age (dynamic/trace! "age" dist/uniform-discrete 18 100)
            likes_video_games (if (< age 60)
                  (dynamic/trace! "likes_video_games" dist/bernoulli 0.9)
                  (dynamic/trace! "likes_video_games" dist/bernoulli 0.1))]
        [age, likes_video_games]))
    {:variables #{"age" "likes_video_games"}}))

;; Create a harmonized model with emission functions that uses age categories as
;; opposed to integers for age.
(def harmonized-model-with-emission
  (with-meta ;; users have to specify all variables in the Gen model that will be queried.
           (gen
             []
             (let [[age likes_video_games] (dynamic/splice! population-model)
                   age-group-probabilities (fn [age-group]
                                             (assoc {"18-30" 0.0
                                                     "31-40" 0.0
                                                     "41-50" 0.0
                                                     "51-60" 0.0
                                                     "60+"   0.0}
                                                    age-group
                                                    1.0))]
             (cond
              (<= age 30)
              (dynamic/trace! "age_group" dist/categorical (age-group-probabilities "18-30"))

              (<= age 40)
              (dynamic/trace! "age_group" dist/categorical (age-group-probabilities "31-40"))

              (<= age 50)
              (dynamic/trace! "age_group" dist/categorical (age-group-probabilities "41-50"))

              (<= age 60)
              (dynamic/trace! "age_group" dist/categorical (age-group-probabilities "51-60"))

              :else
              (dynamic/trace! "age_group" dist/categorical (age-group-probabilities "60+")))))

           {:variables #{"age_group" "likes_video_games"}}))
```

Fig. 17. Gen.clj code for a simple population model and a harmonized model with emission functions.

by caching the results of the likelihood evaluations and the conditioned rowModels in a way that would not typically appear in other general purpose PPLs.

A second class of optimizations exploits the conditional independence relations between subsets of the columns of probabilistic models. We can simplify a query of the form **PROBABILITY OF** c_1 **UNDER** ID

GIVEN $c_2 \wedge \text{ID}.y\ op\ e$ to PROBABILITY OF c_1 UNDER ID GIVEN c_2 if y is independent from the joint distribution of the variables appearing in c_1 and c_2. This independence check can sometimes be performed by simply looking at the structure of the program. For probabilistic programs produced by CrossCat, we can read that a variable y is independent from variables x, z if y is not in the same cluster as x and z. If we see these programs as Bayesian networks, the reason is that no column-variables have nodes as children, and this is what we exploit in our implementation. More generally, one could use more sophisticated algorithms to detect independence relations, for example in Bayesian networks based on d-separation [59]. However, there may be more independence relations that cannot be directly detected, even in the case of exact models expressed as sum-product expressions, and in that case more independence relations can perhaps be detected by using more sophisticated algorithms.

Together, these optimizations can lead to significant speedups in the evaluation of GenSQL queries. One reason is that these optimizations can synergize with each other. To see this, imagine as a first example if ID GIVEN $c \wedge \text{ID}.y = 3$ and ID GIVEN $c \wedge \text{ID}.y = 4$ both get simplified to ID GIVEN c. Then, caching the first conditioned model will hit for the second query. As a second example, consider a subquery ID GIVEN $\text{ID}.x =$ (PROBABILITY OF $\text{ID}'.y = 3$ UNDER ID') appearing at least twice in a query. If ID' only approximates the computation of the probability query, then it's unlikely for ID' to compute the same approximation twice. In that case, caching the result instead allows to compute only one conditioned model for ID, instead of two.

B.1 Caching

Caching conditioned models. GenSQL is agnostic to the implementation of the AMI methods, as long as they satisfy the interface described in Section 4. In particular, the **simulate**, **logpdf** and **prob** methods are free to construct and cache any intermediate data structures, such as a data structure that represents the conditioned model.

Caching exact* prob *and* logpdf *computations. A cache hit typically occurs in GenSQL queries when scalar expressions e_1 and e_2 in an event reduce to the same value. To see why, let $\Gamma \vdash c_1 := c' \wedge (\text{ID}, i)\ op\ e_1$ and $\Gamma \vdash c_2 := c' \wedge (\text{ID}, i)\ op\ e_2$ be two lowered events that differ only in the expressions e_1 and e_2. Let $\Gamma \vdash \mathbf{prob}_{\text{ID}}(c^0, c^1, c_1)$ and $\Gamma \vdash \mathbf{prob}_{\text{ID}}(c^0, c^1, c_2)$ both be well-typed programs. Also assume $C[]$ is a given program with a hole such that $\Gamma \vdash C[\mathbf{prob}_{\text{ID}}(c^0, c^1, c_1)]$ and $\Gamma \vdash C[\mathbf{prob}_{\text{ID}}(c^0, c^1, c_2)]$ are well-typed. Finally, suppose that for some evaluation γ of the context Γ, $[\![e_1]\!]_{\text{exact}}(\gamma) = [\![e_2]\!]_{\text{exact}}(\gamma)$. Then, we will have $[\![\mathbf{prob}_{\text{ID}}(c^0, c^1, c_1)]\!]_{\text{exact}}(\gamma) = [\![\mathbf{prob}_{\text{ID}}(c^0, c^1, c_2)]\!]_{\text{exact}}(\gamma)$. Therefore, we can cache the result of $\mathbf{prob}_{\text{ID}}(c^0, c^1, c_1)$ and reuse it instead of doing the full computation of $\mathbf{prob}_{\text{ID}}(c^0, c^1, c_2)$. More generally, the following proposition justifies the correctness of caching in GenSQL.

PROPOSITION B.1 (CORRECTNESS OF CACHING (EXACT COMPUTATIONS)). *Let P be either* $\mathbf{prob}_{ID}$ *or* $\mathbf{logpdf}_{ID}$. *Let $C[]$ be a program with a hole such that $\Gamma \vdash C[P(c^0, c^1, c)]$ is well-typed. Let γ be an evaluation of the context Γ, and let $v := [\![c]\!]_{exact}(\gamma)$. Then, $\Gamma \vdash C[v]$ is well-typed and $[\![C[P(c^0, c^1, c)]]\!]_{exact}(\gamma) = [\![C[v]]\!]_{exact}(\gamma)$.*

PROOF. This follows by a straightforward induction on the structure of the program with a hole $C[]$. If $C[] := []$, then the result follows by the definition of the semantics and the fact that reals constants v are in the language. Otherwise, $C[]$ is obtained using a rule from the type system in Fig. 20. By induction hypothesis, all the subprograms of $C[]$ are well-typed and the result holds for them. Therefore, the typing result holds for $C[]$. In addition, by compositionality of the semantics, the semantic equality also holds for $C[]$. □

If we take $v := [\![P(c^0, c^1, c_1)]\!]_{\text{exact}}(\gamma) = [\![P(c^0, c^1, c_2)]\!]_{\text{exact}}(\gamma)$ in the above proposition, then

$$[\![C[P(c^0, c^1, c_1)]]\!]_{\text{exact}}(\gamma) = [\![C[v]]\!]_{\text{exact}}(\gamma) = [\![C[P(c^0, c^1, c_2)]]\!]_{\text{exact}}(\gamma)$$

and we have formally recovered the example from above with events differing in a scalar e_1 and e_2. This is a view on caching as partial evaluation.

Caching of approximate* prob *and* logpdf *computations. In the case where prob or logpdf are approximated, the argument using the reduction to a value v above does not directly hold. Indeed, in general we can have $[\![c_1]\!]_{\text{exact}}(\gamma) = [\![c_2]\!]_{\text{exact}}(\gamma)$ but $[\![c_1]\!]_{\text{approx}}(\gamma) \neq [\![c_2]\!]_{\text{approx}}(\gamma)$. Caching in this case *does change* the semantics of the program, but it will not change the asymptotic guarantees of the program.

The key intuition that makes it hold is that if two sequences of random variables $(x_n)_n$ and $(y_n)_n$ converge to the same x, then for every continuous function f, both sequences $(f(x_n, y_n))_n$ and $(f(x_n, x_n))_n$ will converge to $f(x, x)$. Here, x_n and y_n are two approximations of the same quantity that appears twice in a program. The value x is the true value of the quantity under consideration, and the denotation of the program is $f(x_n, y_n)$. With this view, caching consists of using the same approximation x_n twice instead of independently recomputing an approximation y_n. This operation is then valid as long as programs are continuous at (x, x), which is the case for *safe* (see Appendix D.4) programs in GenSQL. In other words, the caching operation may change the approximate semantics but not the asymptotic guarantees of the program, as long as the replaced approximation sequence converges to the same value, and that the query using these approximations is safe.

Proposition B.2 (Correctness of caching (approximate computations)). *Suppose that the following four queries are safe: $\Gamma; \Delta \vdash t_1$, $\Gamma; \Delta \vdash t_2$, $\Gamma; [] \vdash C[t_1, t_2]$ and $\Gamma; [] \vdash C[t_1, t_1]$. Let γ be an evaluation context for Γ, and δ, δ' the*

obtained evaluations of Δ for t_1, t_2 when evaluating $C[t_1, t_2]$ in γ. Further assume that $\delta = \delta'$, and that almost surely

$$\lim_{n\to\infty} [\![\mathcal{T}_\delta\{t_1\}]\!]_{approx}(\gamma)_n = \lim_{n\to\infty} [\![\mathcal{T}_\delta\{t_2\}]\!]_{approx}(\gamma).$$

Then, almost surely

$$\lim_{n\to\infty} [\![\mathcal{T}_{[]}\{C[t_1, t_2]\}]\!]_{approx}(\gamma)_n = \lim_{n\to\infty} [\![\mathcal{T}_{[]}\{C[t_1, t_1]\}]\!]_{approx}(\gamma)_n$$

Proof. Almost surely,

$$\begin{aligned}
&\lim_n [\![\mathcal{T}_{[]}\{C[t_1, t_2]\}]\!]_{\text{approx}}(\gamma)_n \\
&= [\![C[t_1, t_2]]\!](\gamma, []) && \text{Theorem 4.3 as } C[t_1, t_2] \text{ safe} \\
&= [\![C]\!][[\![t_1]\!](\gamma, \delta), [\![t_2]\!](\gamma, \delta)] && \text{Compositionality of } [\![-]\!] \\
&= [\![C]\!][[\![t_1]\!](\gamma, \delta), \lim_{n\to\infty} [\![\mathcal{T}_\delta\{t_2\}]\!]_{\text{approx}}(\gamma)] && \text{Theorem 4.3 as } t_2 \text{ safe} \\
&= [\![C]\!][[\![t_1]\!](\gamma, \delta), \lim_{n\to\infty} [\![\mathcal{T}_\delta\{t_1\}]\!]_{\text{approx}}(\gamma)] && \text{Hypothesis on } t_1, t_2 \\
&= [\![C]\!](\gamma, [])[[\![t_1]\!](\gamma, \delta), [\![t_1]\!](\gamma, \delta)] && \text{Theorem 4.3 as } t_1 \text{ safe} \\
&= [\![C[t_1, t_1]]\!](\gamma, []) && \text{Compositionality of } [\![-]\!] \\
&= \lim_n [\![\mathcal{T}_{[]}\{C[t_1, t_1]\}]\!]_{\text{approx}}(\gamma)_n && \text{Theorem 4.3 as } C[t_1, t_1] \text{ safe}
\end{aligned}$$

□

B.2 Independence simplification

The formal correctness of independence simplification hinges on the repeated application of the following proposition. For ease of expression, we use the following notation in the statement and proof of the proposition.

- The symbols $\bar{i}$ and $\bar{j}$ are respectively syntactic sugar for the sequences $i_1, \dots, i_k$ and $j_1, \dots, j_k$. Similarly, $x_{\bar{i}}$ is sugar for the sequence $x_{i_1}, \dots, x_{i_k}$, and $\pi_{\bar{i}}$ is sugar for $\pi_{i_1} \otimes \cdots \otimes \pi_{i_k}$. We also take $[\![\sigma_{\bar{i}}]\!]$ as sugar for the product $[\![\sigma_{i_1}]\!] \times \cdots \times [\![\sigma_{i_k}]\!]$ and $[\![t_{\bar{i}}]\!](\gamma, \delta)$ as sugar for the sequence $[\![t_1]\!](\gamma, \delta), \dots, [\![t_k]\!](\gamma, \delta)$.
- Given $m \in [\![M[\text{ID}]\{\text{COLS}\}]\!]$, we let
$$p_{\bar{i}|\bar{j}}(x_{\bar{i}}|x_{\bar{j}}) = \mathbf{Dis}(m, \pi_{\bar{j}}, x_{\bar{j}}).\text{pdf}(x_{\bar{i}}).$$
- As in Section 3.1, COLS is sugar for $\text{COL}_1 : \sigma_1, \dots, \text{COL}_n : \sigma_n$. We additionally use $\text{COLS}_{\bar{i}}$ as sugar for $\text{COL}_{i_1} : \sigma_{i_1}, \dots, \text{COL}_{i_k} : \sigma_{i_k}$.

Proposition B.3 (Correctness of independence simplification). *Suppose $m \in [\![M[\mathit{ID}]\{\mathit{COLS}\}]\!]$ and that under m.meas the distributions of $\mathit{COLS}_{\bar{i}}$ and COL_j are conditionally independent given $\mathit{COLS}_{\bar{j}}$; i.e. $p_{\bar{i},j|\bar{j}}(x_{\bar{i}}, x_j|x_{\bar{j}}) = p_{\bar{i}|\bar{j}}(x_{\bar{i}}|x_{\bar{j}})$, or equivalently, for all measurable functions $f : [\![\sigma_{\bar{i}}]\!] \to \mathbb{R}$ and $g : [\![\sigma_j]\!] \to \mathbb{R}$, we have*
$$\int f(x_{\bar{i}})g(x_j)p_{\bar{i},j|\bar{j}}(x_{\bar{i}}, x_j|x_{\bar{j}})dx_{\bar{i}}dx_j = \left(\int f(x_{\bar{i}})p_{\bar{i}|\bar{j}}(x_{\bar{i}}|x_{\bar{j}})dx_{\bar{i}}\right)\left(\int g(x)p_{j|\bar{j}}(x|x_{\bar{j}})dx\right).$$
Let $c = \mathit{ID}.\mathit{COL}_{i_1}\, op_1 t_1 \wedge \cdots \wedge \mathit{ID}.\mathit{COL}_{i_k}\, op_k t_k$, and let (Γ, Δ) be a pair of contexts such that $\Gamma; \Delta \vdash c_0 : C^0\{\mathit{COLS}_{\bar{j}}\}$, and $\Gamma, \Delta \vdash c_1 : C^1\{\mathit{COLS}\}$. Further, let $(\gamma, \delta) \in [\![\Gamma]\!] \times [\![\Delta]\!]$ such that $[\![\mathit{ID}]\!](\gamma, \delta) = \mu$. If $\mathit{COL}_j \notin \mathbf{vars}(c_0) \cup \mathbf{vars}(c_1)$ and
$$\Gamma; \Delta \vdash (\ \textit{PROBABILITY OF } c \textit{ UNDER } \mathit{ID} \textit{ GIVEN } (\mathit{ID}.\mathit{COL}_j\ op'\ t') \wedge c_0 \textit{ GIVEN } c_1) : \tau$$
for some $\tau \in \{\textbf{PosReal}, \textbf{Ranged}(0, 1)\}$, then
$$[\![\ \textit{PROBABILITY OF } c \textit{ UNDER } \mathit{ID} \textit{ GIVEN } (\mathit{ID}.\mathit{COL}_j\ op'\ t') \wedge c_0 \textit{ GIVEN } c_1]\!](\gamma, \delta) =$$
$$[\![\ \textit{PROBABILITY OF } c \textit{ UNDER } \mathit{ID} \textit{ GIVEN } c_0 \textit{ GIVEN } c_1]\!](\gamma, \delta).$$

Proof. For ease of notation suppose $c_0 = \text{ID}.\text{COL}_{j_1} = t'_1 \wedge \cdots \wedge \text{ID}.\text{COL}_{j_l} = t'_l$. We have two cases:

Case I: $c : C^0$. In this case we have
$$[\![\ \textbf{PROBABILITY OF } c \textbf{ UNDER } \text{ID} \textbf{ GIVEN } (\text{ID}.\text{COL}_j\ op'\ t') \wedge c_0 \textbf{ GIVEN } c_1]\!](\gamma, \delta)$$
$$= \frac{1}{\mu([\![c_1]\!](\gamma, \delta))} \prod_{\alpha=1}^{k} \mathbf{1}_{\pi_{i_\alpha}([\![c_1]\!](\gamma,\delta))}([\![t_\alpha]\!](\gamma, \delta)) \prod_{\beta=1}^{l} \mathbf{1}_{\pi_{j_\beta}([\![c_1]\!](\gamma,\delta))}([\![t'_\beta]\!](\gamma, \delta))$$
$$\times p_{\bar{i}|j,\bar{j}}([\![t_{\bar{i}}]\!](\gamma, \delta)|[\![t']\!](\gamma, \delta), [\![t'_{\bar{j}}]\!](\gamma, \delta))$$
$$= \frac{1}{\mu([\![c_1]\!](\gamma, \delta))} \prod_{\alpha=1}^{k} \mathbf{1}_{\pi_{i_\alpha}([\![c_1]\!](\gamma,\delta))}([\![t_\alpha]\!](\gamma, \delta)) \prod_{\beta=1}^{l} \mathbf{1}_{\pi_{j_\beta}([\![c_1]\!](\gamma,\delta))}([\![t'_\beta]\!](\gamma, \delta))$$
$$\times p_{\bar{i}|\bar{j}}([\![t_{\bar{i}}]\!](\gamma, \delta)|[\![t'_{\bar{j}}]\!](\gamma, \delta)) \quad \text{by conditional independence}$$
$$= [\![\ \textbf{PROBABILITY OF } c \textbf{ UNDER } \text{ID} \textbf{ GIVEN } c_0 \textbf{ GIVEN } c_1]\!](\gamma, \delta).$$

Case II: $c : C^1$. Let
$$A_\alpha := \left\{x \in [\![\sigma_{i_\alpha}]\!] \middle| [\![op_\alpha]\!](\gamma, \delta)(x, [\![t_\alpha]\!](\gamma, \delta)) = \textbf{true}\right\} \qquad 1 \le \alpha \le k$$
$$A := A_1 \times \cdots \times A_k.$$

We have

$$\begin{aligned}
&[\![\textbf{PROBABILITY OF } c \textbf{ UNDER } \mathrm{ID} \textbf{ GIVEN } (\mathrm{ID.COL}_j\ op'\ t') \wedge c_0 \textbf{ GIVEN } c_1]\!](\gamma, \delta) \\
&\quad = \frac{1}{\mu([\![c_1]\!](\gamma,\delta))} \int \prod_{\alpha=1}^{k} \mathbf{1}_{\pi_{i_\alpha}([\![c_1]\!](\gamma,\delta))}([\![t_\alpha]\!](\gamma,\delta)) \prod_{\beta=1}^{l} \mathbf{1}_{\pi_{j_\beta}([\![c_1]\!](\gamma,\delta))}([\![t'_\beta]\!](\gamma,\delta)) \\
&\qquad\qquad \times \mathbf{1}_A(x_{\bar{i}}) p_{\bar{i}|j,\bar{j}}(x_{\bar{i}} | [\![t']\!](\gamma,\delta), [\![t'_{\bar{j}}]\!](\gamma,\delta)) dx_{\bar{i}} \\
&\quad = \frac{1}{\mu([\![c_1]\!](\gamma,\delta))} \int \prod_{\alpha=1}^{k} \mathbf{1}_{\pi_{i_\alpha}([\![c_1]\!](\gamma,\delta))}([\![t_\alpha]\!](\gamma,\delta)) \prod_{\beta=1}^{l} \mathbf{1}_{\pi_{j_\beta}([\![c_1]\!](\gamma,\delta))}([\![t'_\beta]\!](\gamma,\delta)) \\
&\qquad\qquad \times \mathbf{1}_A(x_{\bar{i}}) p_{\bar{i}|\bar{j}}(x_{\bar{i}} | [\![t'_{\bar{j}}]\!](\gamma,\delta)) dx_{\bar{i}} \quad \text{by conditional independence} \\
&\quad = [\![\textbf{PROBABILITY OF } c \textbf{ UNDER } \mathrm{ID} \textbf{ GIVEN } c_0 \textbf{ GIVEN } c_1]\!](\gamma,\delta).
\end{aligned}$$

□

C IMPLEMENTATIONS OF THE AMI

C.1 AMI in terms of sum-product expressions

Sum-product networks (SPNs) are a useful family of distributions that are closed under marginalization and conditioning. Sum-product expressions (SPEs) [73] generalize SPNs to gain more expressivity while still allowing for exact marginalization and conditioning. The idea is that if finitely many distributions are closed under marginalization and conditioning, so are their independent products and weighted mixtures. On top of having an API method for sampling, SPEs also have the API methods **prob** for computing the probability of an event and **logpdf** for computing the (marginal) log-density of the distribution represented by the SPE. SPEs also have access to the following functions, which are typically implemented as program transforms:

$$\begin{aligned}
\textbf{cond}_0 &: C^0\{\textsc{cols}'\} \to \mathrm{SPE} \to \mathrm{SPE} \\
\textbf{cond}_1 &: C^1\{\textsc{cols}\} \to \mathrm{SPE} \to \mathrm{SPE} \\
\textbf{marginalize}_{\textsc{cols}'} &: \mathrm{SPE} \to \mathrm{SPE}.
\end{aligned}$$

The functions $\textbf{cond}_0$ and $\textbf{cond}_1$ take in an SPE and an event-0 or event, respectively, and return a new SPE representing the conditional distribution of the input SPE given the provided event or event-0. The family of methods $\textbf{marginalize}_{\textsc{cols}'}$ take an SPE and return a new SPE whose distribution is the marginal distribution of $\textsc{cols}'$ under the input SPE. They can for instance be used to implement marginal densities in terms of densities.

With these, we can implement the AMI methods as follows:

$$\begin{aligned}
\textbf{simulate}_{\mathrm{ID}}(c^0, c^1) &= \textbf{sample}\ \$\ \textbf{cond}_0\ c^0\ \$\ \textbf{cond}_1\ c^1\ M_{\mathrm{ID}} \\
\textbf{logpdf}_{\mathrm{ID}}(c^0, c^1, c^0_2) &= \textbf{logpdf}\ c^0_2\ \$\ \textbf{cond}_0\ c^0\ \$\ \textbf{cond}_1\ c^1\ M_{\mathrm{ID}} \\
\textbf{prob}_{\mathrm{ID}}(c^0, c^1, c^1_2) &= \textbf{prob}\ c^1_2\ \$\ \textbf{cond}_0\ c^0\ \$\ \textbf{cond}_1\ c^1\ M_{\mathrm{ID}}.
\end{aligned}$$

C.2 AMI in terms of truncated multivariate Gaussians

It is well-known that the family of multivariate Gaussian distributions is closed under conditioning on a subset of the Gaussian variates. Moreover, if $\mathbf{x} \sim \mathcal{N}(\mu, \Sigma)$ is a d-dimensional multivariate Gaussian random variable, A is a $m \times d$ matrix, and $\mathbf{u}, \mathbf{l} \in \mathbb{R}^m$, then conditional on the event $\{\mathbf{l} \leq A\mathbf{x} \leq \mathbf{u}\}$ the random variable $\mathbf{x}$ follows a truncated multivariate Gaussian distribution.

If we focus on the set of events of the form $\{\mathbf{l} \leq A\mathbf{x} \leq \mathbf{u}\}$, then similar to sum-product expressions, the family of truncated multivariate Gaussians will be closed under conditioning. This results from the fact that multivariate Gaussians are closed under conditioning on event-0s and conditioning a truncated multivariate Gaussian on an event of the above form is equivalent to truncating it further. That is, if we represent a truncated multivariate Gaussian by the tuple $(\mu, \Sigma, A, \mathbf{l}, \mathbf{u})$, we can define the events

$$\mathbf{cond} : C^0\{\textsc{cols}'\} \to \text{TMVG} \to \text{TMVG}$$
$$\mathbf{truncate} : C^1\{\textsc{cols}\} \to \text{TMVG} \to \text{TMVG},$$

where TMVG denotes the type of truncated multivariate Gaussian terms.

Moreover, it is possible to define distribution functions for multivariate Gaussians distributions allowing us to calculate the probability mass assigned to hyperrectangles in the domain of the distribution. We note that such calculations require the evaluation of the distribution function at every corner of the hyperrectangles, and the number of these corners can be exponential in the dimension of the hyperrectangle. Furthermore, there are several exact methods for exactly calculating the normalizing constant of truncated multivariate Gaussian variables. We refer the reader to the textbook [31] for the details, and assume access to methods **logpdf** and **prob** for calculating log-marginal densities and probabilities, respectively.

Using the functions **truncate**, **cond**, **logpdf**, and **prob** the implementation of the AMI methods for truncated multivariate Gaussians is almost identical to that of SPEs, as follows.

$$\mathbf{simulate}_{\textsc{id}}(c^0, c^1) = \mathbf{sample}\ \$\ \mathbf{cond}_0\ c^0\ \$\ \mathbf{truncate}\ c^1\ M_{\textsc{id}}$$
$$\mathbf{logpdf}_{\textsc{id}}(c^0, c^1, c^0_2) = \mathbf{logpdf}\ c^0_2\ \$\ \mathbf{cond}\ c^0\ \$\ \mathbf{truncate}\ c^1\ M_{\textsc{id}}$$
$$\mathbf{prob}_{\textsc{id}}(c^0, c^1, c^1_2) = \mathbf{prob}\ c^1_2\ \$\ \mathbf{cond}\ c^0\ \$\ \mathbf{truncate}\ c^1\ M_{\textsc{id}}.$$

C.3 AMI in terms of ancestral sampling

In general, exact Bayesian inference in probabilistic models is intractable. As such, practitioners often rely on variational or Monte Carlo methods for approximate inference. Probabilistic programming languages that support programmable inference allow their users to seamlessly incorporate approximate inference methods into their workflow. We now describe how GenSQL's AMI can be implemented in such probabilistic programming languages. As a simple representative example of an approximate inference algorithm we chose ancestral sampling, a simple sequential Monte Carlo method.

We do not introduce ancestral sampling in detail here, and refer the unfamiliar reader to [64]. As in the previous sections, we describe our implementation in a superset of the lowered language. We assume the implementation language contains a parametric type Model{cols} representing a probabilistic model describing a joint distribution on the columns cols. Given this type, the type signature of the ancestral sampling algorithm is given by

$$\mathbf{ancestral} : \text{Model}\{\textsc{cols}\} \to C^0\{\textsc{cols}'\} \to C^1\{\textsc{cols}\} \to ((\sigma_1, \ldots, \sigma_n), \mathbf{Real}).$$

In the above type signature, the first argument denotes the probabilistic model under consideration. The remaining arguments denote the event-0 and event on which the model is being conditioned. The algorithm returns a weighted sample (x, w), where w is the log-importance weight of x.

Recall that the approximate implementations of the AMI methods are indexed by a "compute budget" n. Here, n will denote the number of independent ancestral samples we will generate to perform the desired computation.

Throughout, we suppose $M_{\textsc{id}}$ is a probabilistic model in the host language implementing the row model specified by id.

To implement $\mathbf{simulate}_{\textsc{id}}(c^0, c^1)$, we first generate n independent ancestral samples by calling **ancestral** $M_{\textsc{id}}$ c^0 c^1 for n times. If the importance weights of all the generated particles are zero (i.e. the return second element of all the returned tuples are $-\infty$) then the generated sample is incompatible with the event described by c^1. As such, in this case we return a null sample $(\star, \dots, \star)$. Note that as the number of particles n increases, the probability of generating such a collection of particles goes to zero, and so the implementation of simulate will generate a non-null sample almost surely. On the other hand, if at least one importance weight is non-zero, we re-sample one of the generated particles based on its weight and return it. The sampling distribution of the return value of this procedure converges to the conditioned row model by an elementary statistical argument [64]. The Haskell-style pseudo-code of the **simulate** method is given below.

```
$\textbf{simulate}_{\text{ID}}(c^0, c^1) = \textbf{let}$ particles = $\textbf{replicate}\ n\ \$\ \textbf{ancestral}\ M_{\text{ID}}\ c^0\ c^1$ in
                    let logits = map $\pi_2$ particles in
                      if all (== $-\infty$) logits then $(\star, \dots, \star)$ else
                        let $i$ = Categorical \$ logitsToProbs logits in
                          $\pi_1$ particles$_i$
```

To implement **logpdf**(c^0, c^1, c_2^0) we rely on the fact that the expected value of an importance weight is equal to the joint density of the variables whose values are given [47]. Using this fact, we first define a helper function **logmarginal** which estimates the log marginal density of a given assignment of variables to values expressed as an event-0. That is, we define

$$\textbf{logmarginal} : \text{Model}\{\text{COLS}\} \to C^0\{\text{COLS}'\} \to C^1\{\text{COLS}\} \to \textbf{Real},$$

$$\textbf{logmarginal}\ M\ c^0\ c^1 = \textbf{logmeanexp}\ \$\ \textbf{map}\ \pi_2\ (\textbf{replicate}\ n\ \$\ \textbf{ancestral}\ M\ c^0\ c^1)$$

where **logmeanexp** xs := (**logsumexp** xs) − (log \$ **length** xs). Now, as conditional densities are given as a ratio of marginal densities we will have the implementation of $\textbf{logpdf}_{\text{ID}}(c^0, c^1, c_2^0)$ as follows

$$\textbf{logpdf}_{\text{ID}}(c^0, c^1, c_2^0) = (\textbf{logmarginal}\ M_{\text{ID}}\ (c^0 \wedge c_2^0)\ c^1) - (\textbf{logmarginal}\ M_{\text{ID}}\ c^0\ c^1).$$

Note that as $n \to \infty$, the return value of this implementation will almost surely converge to the correct value. This is because by the strong law of large numbers the results of **logmarginal** will almost surely converge to the correct value, and as division and logarithm are continuous functions the value of $\textbf{logpdf}_{\text{ID}}(c^0, c^1, c_2^0)$ will converge to the correct value.

Lastly, to implement the **prob** method we rely on the fact that we can perform Monte Carlo integration using importance samples. Note that the probability of an event under a probability measure is given by the integral of the indicator function of that event under the same measure. We have the implementation

$$\begin{aligned}\textbf{prob}_{\text{ID}}(c^0, c^1, c_2^1) = \textbf{let}\ &\text{particles} = \textbf{replicate}\ n\ \$\ \textbf{ancestral}\ M_{\text{ID}}\ c^0\ c^1\ \textbf{in}\\ &\exp\big(\textbf{logmeanexp}\ [w \times (\textbf{logindicator}\ c^1\ x) | (x, w) \in \text{particles}]\\ &\quad - \textbf{logmarginal}\ M_{\text{ID}}\ c^0\ c^1\big),\end{aligned}$$

where **logindicator** $c^1\ x$:= **if** $x \in c^1$ **then** 0 **else** $-\infty$.

Note that we need the correction term **logmarginal** $M_{\text{ID}}\ c^0\ c^1$, as Monte Carlo integral estimators obtained via ancestral sampling estimate the integral under consideration up to a constant factor which is the joint density of the given variables.

D LOWERING DETAILS

D.1 Normalization of GenSQL Queries

Our implementation allows arbitrary nested conditionings on events and event-0 which do not fit the restriction from the type system presented in Section 3. Our normalization has two stages (Figs. 18 and 19), which rewrite queries so they satisfy the formalization presented in Section 3, and simplifies the queries to a normal form.

The first part of the normalization does three things.

(1) It partially evaluates RENAME clauses on models.
(2) It aggregates events in GIVEN clauses.
(3) It removes self-contradictory statements. For instance, PROBABILITY OF $\text{ID}.x = 7 \wedge \text{ID}.y = 8$ UNDER ID GIVEN $\text{ID}.y = 4$ should technically return 0 as the event is impossible, but the normalization will return PROBABILITY OF $\text{ID}.x = 7$ UNDER ID GIVEN $\text{ID}.y = 4$ instead. These can also appear through nested GIVEN clauses, e.g. (ID GIVEN $\text{ID.COL}_1 = 7$) GIVEN $\text{ID.COL}_1 = 8$. In this case, the normalization pass will remove the second GIVEN clause. In other words, variables which are conditioned by a 0-event behave in the density of the model like variables which have been marginalized-out and are not present in the model anymore. The same logic applies to the probability of an event under a model, but this case does not need a special treatment. Ideally, an implementation should issue a warning to the user letting them know that there was probably a mistake in the query.

The second part of the normalization pass is only needed for the implementation and ensures that we do not evaluate a density at a point where some of the indices have been conditioned on. This avoids another subtle issue when conditioning on event-0. Briefly, when conditioning on an event-0 such as ID.COL = 7, we are changing the base measure on COL to be a Dirac δ measure at 7, and the density evaluation would incorrectly assume that it is still the original base measure. In the second part of the normalization, a temporary name **true** is used which is not part of the language syntax. It gets eliminated during the normalization. After applying the normalization passes, we obtain the following normal forms:

Proposition D.1. *The rewrites from Fig. 18 are confluent, terminating, and lead to the following normal forms, where RENAME and GIVEN clauses are optional:*

- *rowModels:* RENAME $(\text{ID}$ GIVEN c^0 GIVEN $c^1)$ AS ID'.
- *Probability queries:* PROBABILITY OF c_1^i UNDER $(\text{ID}$ GIVEN c^0 GIVEN $c^1)$.
- *Generate queries:* GENERATE UNDER $(\text{ID}$ GIVEN c^0 GIVEN $c^1)$ LIMIT e *and* t GENERATIVE JOIN $(\text{ID}$ GIVEN c^0 GIVEN $c^1)$.

After the rewrites from Fig. 19, we can further assume that the variables in c_1^0 and c^0 are disjoint.

Proof. Termination: we define the following valuation function **val** on expressions:

- $\mathbf{val}\,(\text{ID}) = 1$
- $\mathbf{val}\,($ RENAME m AS $\text{ID}') = 1 + \mathbf{val}\,(m)$
- $\mathbf{val}\,(m$ GIVEN $c^0) = 2 * \mathbf{val}\,(m)$
- $\mathbf{val}\,(m$ GIVEN $c^1) = 2 * \mathbf{val}\,(m) + 1$
- $\mathbf{val}\,($ PROBABILITY OF c^i UNDER $m) = \mathbf{val}\,(m)$
- $\mathbf{val}\,($ GENERATE UNDER m LIMIT $e) = \mathbf{val}\,(m)$
- $\mathbf{val}\,(t$ GENERATIVE JOIN $m) = \mathbf{val}\,(m)$

Every rewrite rule strictly decreases the valuation of the expression, and thus the rewrite system terminates.

(**RENAME** m **AS** ID$'$) **GIVEN** c^i	$\rightsquigarrow$	**RENAME** (m **GIVEN** $c^i[\text{ID}'/\text{ID}]$) **AS** ID$'$	
RENAME (**RENAME** m **AS** ID$'$) **AS** ID$''$	$\rightsquigarrow$	**RENAME** m **AS** ID$''$	
(m **GIVEN** c^0) **GIVEN** $c^{0'}$	$\rightsquigarrow$	m **GIVEN** $S(c^0, c^{0'})$	
(m **GIVEN** c^1) **GIVEN** $c^{1'}$	$\rightsquigarrow$	m **GIVEN** $c^1 \wedge c^{1'}$	
(m **GIVEN** c^1) **GIVEN** c^0	$\rightsquigarrow$	(m **GIVEN** c^0) **GIVEN** c^1	
PROBABILITY OF c^i **UNDER** (**RENAME** m **AS** ID$'$)	$\rightsquigarrow$	**PROBABILITY OF** $c^i[\text{ID}'/\text{ID}]$ **UNDER** m	
GENERATE UNDER (**RENAME** m **AS** ID) **LIMIT** e	$\rightsquigarrow$	**GENERATE UNDER** m **LIMIT** e	
t **GENERATIVE JOIN** (**RENAME** m **AS** ID)	$\rightsquigarrow$	t **GENERATIVE JOIN** m	
$S(c^0, [])$	$\rightsquigarrow$	c^0	
$S(c^0 : C^0\{\text{COLS}\}, (\text{ID.COL} = e) \wedge c^{0'})$	$\rightsquigarrow$	$S(c^0 \wedge (\text{ID.COL} = e), c^{0'})$	if COL $\notin$ COLS
		$S(c^0, c^{0'})$	otherwise

Fig. 18. Normalization rules for rowModels (first phase).

PROBABILITY OF c_2^0 **UNDER** (ID **GIVEN** c^0 **GIVEN** c^1)	$\rightsquigarrow$	**PROBABILITY OF** $P(c_2^0, c^0)$ **UNDER** ID **GIVEN** c^0 **GIVEN** c^1	
$P(c_2^0 \wedge (\text{ID.COL} = e), c^0 : C^0\{\text{COLS}\})$	$\rightsquigarrow$	$P(c_2^0, c^0)$	if COL $\in$ COLS
		$P(c_2^0, c^0) \wedge (\text{ID.COL} = e)$	otherwise
$P([], c^0)$	$\rightsquigarrow$	**true**	
true $\wedge\, c^0$	$\rightsquigarrow$	c^0	
PROBABILITY OF true UNDER m	$\rightsquigarrow$	1	

Fig. 19. Normalization rules for rowModels (second phase).

Confluence (sketch): this stems from the fact that the rewrites are generated from all critical pairs and the rules for the self-critical pairs are associative, which can be checked by inspection.

Disjointness (sketch): a straightforward induction on the normalization rules including P show that the variables in c_1^0 and c^0 are disjoint after P is applied. □

Figure 20 shows the full syntax of the lowered language. Figure 21 shows the measure semantics of the lowered language for rowModels satisfying the exact AMI.

D.2 Lowered language

Fig. 21 shows the full syntax of the lowered language. Compared to the abbrieviated syntax in Fig. 6, we have added the builtin functions. Fig. 21 presents the exact semantics of the lowered language. The map $l : \mathcal{P}\mathbf{Bag}\,X \to \mathbf{Bag}\,\mathcal{P}X$ in the semantics of **mapreduce** is the distributive law for the point process monad [21]. The semantics of builtin functions is their standard mathematical counterpart.

D.3 Proof of the exact guarantee theorem

First note that the interpretation of the contexts Γ in GenSQL and the lowered language for closed expressions $\Gamma, [] \vdash t : \tau$ are the same, i.e. $[\![\mathcal{T}\{\Gamma\}]\!]_{\text{exact}} = [\![\Gamma]\!]$. This is checked by direct inspection. Given a local context Δ and an evaluation δ of that context, we write $\delta' = \mathcal{T}\{\delta\}$ to denote the corresponding evaluation of the lowered context $\mathcal{T}\{\Delta\}$. $\mathcal{T}\{\delta\}$ is defined as the identity on the table keys, and removes model keys. For general expressions $\Gamma; \Delta \vdash t : \tau$, the context Γ of the lowered and GenSQL program is not necessarily the same. In this case, $\mathcal{T}_{\delta_1}\{t\}$ will be in a context Γ' that consists of $\mathcal{T}\{\Gamma\}$ as well as a new variable for each table in Δ. That is, Γ' adds a renamed version of each table type in Δ to $\mathcal{T}\{\Gamma\}$. Likewise, an evaluation context γ, δ will translate to an evaluation context

$$\text{base type } \sigma ::= \sigma_c \mid \sigma_d \quad \text{ground type } \sigma_g ::= \sigma \mid (\sigma_1, \dots, \sigma_n) \qquad \text{event type } \mathcal{E} ::= C^1[\sigma_g] \mid C^0[\sigma_g]$$

$$\text{type } \tau ::= \mathbf{Bag}[\sigma_g] \quad \text{operator } op ::= + \mid - \mid \times \mid \div \mid \wedge \mid \vee \mid = \quad \text{rowModel } \mathcal{M} ::= M[\sigma_g]$$

$$\text{builtin function } f ::= \mathbf{mapreduce} \mid \mathbf{map} \mid \mathbf{filter} \mid \mathbf{replicate} \mid \mathbf{exp} \mid \mathbf{singleton} \mid \mathbf{join}$$

$$\text{special primitives } ::= \mathbf{simulate} \mid \mathbf{prob} \mid \mathbf{logpdf}$$

$$\text{term } t ::= c \mid \textsc{id} \mid f(t_1, \dots, t_n) \mid x \mid (t_1, \dots, t_n) \mid \pi_i\, t \mid t_1\ op\ t_2$$

$$\frac{\Gamma \vdash t : \sigma_i}{\Gamma, \textsc{id} : M[(\sigma_1, \dots, \sigma_n)] \vdash (\textsc{id}, i) = t : C^0[\sigma_i]} \qquad \frac{\Gamma \vdash t_1 : C^0[\sigma_g^1] \quad \Gamma \vdash t_2 : C^0[\sigma_g^2]}{\Gamma \vdash t_1 \wedge t_2 : C^0[\sigma_g^1, \sigma_g^2]}$$

$$\frac{\Gamma \vdash t : \sigma_i \quad op \in \{=, <, >\} \quad (\sigma_i, op) \neq (\sigma_c, =)}{\Gamma, \textsc{id} : M[(\sigma_1, \dots, \sigma_n)] \vdash (\textsc{id}, i)\ op\ t : C^1[(\sigma_1, \dots, \sigma_n)]} \qquad \frac{\Gamma \vdash t_1 : C^1[\sigma_g] \quad \Gamma \vdash t_2 : C^1[\sigma_g] \quad op \in \{\wedge, \vee\}}{\Gamma \vdash t_1\ op\ t_2 : C^1[\sigma_g]}$$

$$\frac{\Gamma \vdash t_1 : \mathbf{Nat} \quad \Gamma \vdash t_2 : \mathbf{Bag}[\sigma_g]}{\Gamma \vdash \mathbf{replicate}\ (t_1, t_2) : \mathbf{Bag}[\sigma_g]} \qquad \frac{\Gamma \vdash t_1 : \mathbf{Bag}[(\sigma_1, \dots, \sigma_n)] \quad \Gamma \vdash t_2 : \mathbf{Bag}[(\sigma_{n+1}, \dots, \sigma_{n+m})]}{\Gamma \vdash \mathbf{join}(t_1, t_2) : \mathbf{Bag}[(\sigma_1, \dots, \sigma_{n+m})]}$$

$$\frac{\Gamma, x : \sigma_g^2 \vdash t_1 : \sigma_g^1 \quad \Gamma \vdash t_2 : \mathbf{Bag}[\sigma_g^2]}{\Gamma \vdash \mathbf{map}\ (x.t_1)\ t_2 : \mathbf{Bag}[\sigma_g^1]} \qquad \frac{\Gamma, \textsc{id} : M[\sigma_g] \vdash c^i : C^i[(\sigma_1, \dots, \sigma_n)]}{\Gamma, \textsc{id} : M[\sigma_g] \vdash \mathbf{simulate}_{\textsc{id}}(c^0, c^1) : \mathbf{Bag}[\sigma_g]}$$

$$\frac{}{\Gamma, \textsc{id} : \mathbf{Bag}[\sigma_g] \vdash \textsc{id} : \mathbf{Bag}[\sigma_g]} \qquad \frac{\Gamma, \textsc{id} : M[\sigma_g] \vdash c^i : C^i[\sigma_g] \quad \Gamma, \textsc{id} : M[\sigma_g] \vdash c_1^0 : C^0[\sigma_g]}{\Gamma, \textsc{id} : M[\sigma_g] \vdash \mathbf{logpdf}_{\textsc{id}}(c^0, c^1, c_1^0) : \mathbf{Real}}$$

$$\frac{\Gamma, \textsc{id} : M[\sigma_g] \vdash c^i : C^i[\sigma_g] \quad \Gamma, \textsc{id} : M[\sigma_g] \vdash c_1^1 : C^1[\sigma_g]}{\Gamma, \textsc{id} : M[\sigma_g] \vdash \mathbf{prob}_{\textsc{id}}(c^0, c^1, c_1^1) : \mathbf{Ranged}(0, 1)} \qquad \frac{\Gamma, x : \sigma_g \vdash t_1 : \mathbf{Bool} \quad \Gamma \vdash t_2 : \mathbf{Bag}[\sigma_g]}{\Gamma \vdash \mathbf{filter}\ (x.t_1)\ t_2 : \mathbf{Bag}[\sigma_g]}$$

$$\frac{\Gamma \vdash t : \sigma_g}{\Gamma \vdash \mathbf{singleton}(t) : \mathbf{Bag}[\sigma_g]} \qquad \frac{\Gamma, x : \sigma_g^2 \vdash t_1 : \mathbf{Bag}[\sigma_g^1] \quad \Gamma \vdash t_2 : \mathbf{Bag}[\sigma_g^2]}{\Gamma \vdash \mathbf{mapreduce}\ (x.t_1)\ t_2 : \mathbf{Bag}[\sigma_g^1]} \qquad \frac{\Gamma \vdash t_i : \sigma}{\Gamma \vdash t_1\ op\ t_2 : \sigma}(op : \sigma, \sigma \to \sigma)$$

$$\frac{}{\Gamma \vdash c : \sigma} \qquad \frac{}{\Gamma, x : \tau \vdash x : \tau} \qquad \frac{\Gamma \vdash x : \mathbf{Real}}{\Gamma \vdash \mathbf{exp}(x) : \mathbf{PosReal}} \qquad \frac{\Gamma \vdash t : (\sigma_1, \dots, \sigma_n)}{\Gamma \vdash \pi_i(t) : \sigma_i} \qquad \frac{\Gamma \vdash t_i : \sigma_i \quad i = 1..n}{\Gamma \vdash (t_1, \dots, t_n) : (\sigma_1, \dots, \sigma_n)}$$

Fig. 20. Full syntax and type system of the lowered language.

γ' in the lowered language, where γ' is the same as γ on the variables in Γ, and for every key k at position i in δ_1, $\gamma'(k) = \delta(\delta.\text{key}[i])$. Then, the proof of the theorem is by induction on the structure of the GenSQL program. More precisely, we show the following. Let $\Gamma; \Delta \vdash t : \tau$ be a GenSQL program. Then, for all evaluation of the context γ, δ, we have that $[\![t]\!](\gamma, \delta) = [\![\mathcal{T}_{\delta'}\ \{t\}]\!](\gamma')$, where $\delta' = \mathcal{T}\ \{\delta\}$. With no loss of generality, we simplify the problem by assuming that the queries are normalized, i.e. models m can be assumed to be of the form ID GIVEN c^0 GIVEN c^1, where GIVEN clauses are optional. In addition, we will only cover the case where these clauses are present, as the other cases are immediate simplifications.

Semantics of types and contexts

$$[\![\sigma]\!]_{\text{exact}} = [\![\sigma]\!] \qquad [\![(\sigma_1,\dots,\sigma_n)]\!]_{\text{exact}} = [\![\sigma_1]\!]_{\text{exact}} \times \dots \times [\![\sigma_n]\!]_{\text{exact}} \qquad [\![\mathbf{Bag}[\sigma_g]]\!]_{\text{exact}} = \mathcal{P}\mathbf{Bag}([\![\sigma_g]\!])$$

$$[\![M[\sigma_g]]\!]_{\text{exact}} = \mathcal{P}_{\text{adm}}\left([\![\sigma_g]\!]_{\text{exact}}\right) \qquad [\![\Gamma, x:\tau]\!]_{\text{exact}} = [\![\Gamma]\!]_{\text{exact}} \times [\![\tau]\!]_{\text{exact}} \qquad [\![\,[\,]\,]\!]_{\text{exact}} = 1$$

Semantics of terms

$$[\![(t_1,\dots,t_n)]\!]_{\text{exact}}(\gamma) = ([\![t_1]\!]_{\text{exact}}(\gamma),\dots,[\![t_n]\!]_{\text{exact}}(\gamma))$$

$$[\![\pi_i\ t]\!]_{\text{exact}}(\gamma) = \pi_i([\![t]\!]_{\text{exact}}(\gamma))$$

$$[\![op(t_1,t_2)]\!]_{\text{exact}}(\gamma) = op([\![t_1]\!]_{\text{exact}}(\gamma),[\![t_2]\!]_{\text{exact}}(\gamma))$$

$$[\![\text{ID}]\!]_{\text{exact}}(\gamma) = \gamma(\text{ID})$$

$$[\![\mathbf{simulate}_{\text{ID}}(c^0,c^1)]\!]_{\text{exact}}(\gamma) = \mathbf{let}\ (\pi,v) = [\![c^0]\!]_{\text{exact}}(\gamma)\ \mathbf{in}\ \mathbf{let}\ m = \mathbf{Dis}(\gamma(\text{ID}),\pi,v)\ \mathbf{in}$$
$$\mathbf{let}\ E = [\![c^1]\!]_{\text{exact}}(\gamma)\ \mathbf{in}\ \mathbf{cond}(m,E)$$

$$[\![\mathbf{logpdf}_{\text{ID}}(c^0,c^1,c^0_2)]\!]_{\text{exact}}(\gamma) = \mathbf{let}\ (\pi,v) = [\![c^0_2]\!]_{\text{exact}}(\gamma)\ \mathbf{in}\ \log([\![\mathbf{simulate}_{\text{ID}}(c^0,c^1)]\!]_{\text{exact}}(\gamma).\text{pdf}(v))$$

$$[\![\mathbf{prob}_{\text{ID}}(c^0,c^1,c^1_2)]\!]_{\text{exact}}(\gamma) = [\![\mathbf{simulate}_{\text{ID}}(c^0,c^1)]\!]_{\text{exact}}(\gamma).\text{meas}([\![c^1_2]\!]_{\text{exact}}(\gamma))$$

$$[\![(\text{ID},i)\ op\ t : C^1[\sigma_g]]\!]_{\text{exact}}(\gamma) = \{(x_1,\dots,x_n)\in[\![\sigma_g]\!]_{\text{exact}} \mid x_i\ op_l\ [\![t]\!]_{\text{exact}}(\gamma)\}$$

$$[\![(\text{ID},i) = t : C^0[\sigma_g]]\!]_{\text{exact}}(\gamma) = (\pi_i,[\![t]\!]_{\text{exact}}(\gamma))$$

$$[\![c^1_1 \wedge c^1_2]\!]_{\text{exact}}(\gamma) = [\![c^1_1]\!]_{\text{exact}}(\gamma) \cap [\![c^1_2]\!]_{\text{exact}}(\gamma)$$

$$[\![c^1_1 \vee c^1_2]\!]_{\text{exact}}(\gamma) = [\![c^1_1]\!]_{\text{exact}}(\gamma) \cup [\![c^1_2]\!]_{\text{exact}}(\gamma)$$

$$[\![c^0_1 \wedge c^0_2]\!]_{\text{exact}}(\gamma) = \mathbf{let}\ _{1\le i\le 2}(f_i,v_i) = [\![c^0_i]\!]_{\text{exact}}(\gamma)\ \mathbf{in}\ \left(\lambda x.(f_1(x),f_2(x)),(v_1,v_2)\right)$$

$$[\![\mathbf{mapreduce}\ (x.t_1)\ t_2]\!]_{\text{exact}}(\gamma) = l\left[[\![t_2]\!]_{\text{exact}}(\gamma) \gg\!= \left(\lambda S.\ \left\{\lambda x'.[\![t_1]\!]_{\text{exact}}(\gamma[x\mapsto x'])\ y \,\middle|\, y\in S\right\}\right)\right]$$

$$[\![\mathbf{map}\ (x.t_1)\ t_2]\!]_{\text{exact}}(\gamma) = [\![t_2]\!]_{\text{exact}}(\gamma) \gg\!= \left(\lambda S.\mathbf{return}\ \left\{\lambda x'.[\![t_1]\!]_{\text{exact}}(\gamma[x\mapsto x'])\ y \,\middle|\, y\in S\right\}\right)$$

$$[\![\mathbf{filter}\ (x.t_1)\ t_2]\!]_{\text{exact}}(\gamma) = [\![t_2]\!]_{\text{exact}}(\gamma) \gg\!= \left(\lambda S.\mathbf{return}\ \left\{y\in S \,\middle|\, \lambda x'.[\![t_1]\!]_{\text{exact}}(\gamma[x\mapsto x'])\ y\right\}\right)$$

$$[\![\mathbf{replicate}\ (t_1,t_2)]\!]_{\text{exact}}(\gamma) = \mathbf{let}\ m = [\![t_1]\!]_{\text{exact}}(\gamma)\ \mathbf{in}$$
$$[\![t_2]\!]_{\text{exact}}(\gamma) \gg\!= \lambda\mu.\ \bigotimes_{i=1}^m \mu \gg\!= \left(\lambda(A_1,\dots,A_m).\mathbf{return}\ \cup_{i=1}^m A_i\right)$$

$$[\![\mathbf{exp}(t_1)]\!]_{\text{exact}}(\gamma) = \exp\left([\![t_1]\!]_{\text{exact}}(\gamma)\right)$$

$$[\![\mathbf{singleton}(t_1)]\!]_{\text{exact}}(\gamma) = \mathbf{return}\ \{[\![t_1]\!]_{\text{exact}}(\gamma)\}$$

$$[\![\mathbf{join}(t_1,t_2)]\!]_{\text{exact}}(\gamma) = [\![t_1]\!]_{\text{exact}}(\gamma) \otimes [\![t_2]\!]_{\text{exact}}(\gamma) \gg\!= \left(\lambda S,S'.\mathbf{return}\ (\mathbf{map2\ splat}\ S\ S')\right)$$

Fig. 21. Exact semantics of the lowered language.

- $t \equiv$ GENERATE UNDER $(\text{ID}$ GIVEN c^0 GIVEN $c^1)$ LIMIT e

$$[\![\mathcal{T}_{\delta'}\left\{\text{GENERATE UNDER } (\text{ID GIVEN } c^0 \text{ GIVEN } c^1) \text{ LIMIT } e\right\}]\!]_{\text{exact}}(\gamma')$$
$$= [\![\mathbf{replicate}\ (\mathcal{T}_{\delta'}\{e\}, \mathbf{simulate}_{\text{ID}}(\mathcal{T}_{\delta'}\{c^0\},\mathcal{T}_{\delta'}\{c^1\}))]\!]_{\text{exact}}(\gamma') \qquad \text{def } \mathcal{T}\{-\}$$
$$= \mathbf{let}\ m = [\![\mathcal{T}_{\delta'}\{e\}]\!]_{\text{exact}}(\gamma')\ \mathbf{in} \qquad \text{def } [\![\mathbf{replicate}\,]\!]_{\text{exact}}$$
$$[\![\mathbf{simulate}_{\text{ID}}(\mathcal{T}_{\delta'}\{c^0\},\mathcal{T}_{\delta'}\{c^1\})]\!]_{\text{exact}}(\gamma') \gg\!= \lambda\mu.\ \bigotimes_{i=1}^m \mu \gg\!= \left(\lambda(A_1,\dots,A_m).\mathbf{return}\ \cup_{i=1}^m A_i\right)$$
$$= \mathbf{let}\ m = [\![e]\!](\gamma,\delta)\ \mathbf{in} \qquad \text{I.H.}$$
$$[\![\text{ID GIVEN } c^0 \text{ GIVEN } c^1]\!](\gamma,\delta) \gg\!= \lambda\mu.\ \bigotimes_{i=1}^m \mu \gg\!= \left(\lambda(A_1,\dots,A_m).\mathbf{return}\ \cup_{i=1}^m A_i\right)$$
$$= [\![\text{ GENERATE UNDER } (\text{ID GIVEN } c^0 \text{ GIVEN } c^1) \text{ LIMIT } e]\!](\gamma,\delta) \qquad \text{def } [\![\text{ GENERATE UNDER }]\!]$$

- $t \equiv t_1$ **WHERE** e.

$$
\begin{aligned}
&[\![\mathcal{T}_{\delta'}\ \{t_1\ \textbf{WHERE}\ e\}]\!]_{\text{exact}}(\gamma') \\
&= \big(\lambda x.\textbf{filter}\ (\lambda r.[\![\mathcal{T}_{\delta'[\text{ID}\to r]}\ \{e\}]\!]_{\text{exact}}(\gamma'), x)\big)_*[\![\mathcal{T}_{\delta'}\ \{t_1\}]\!]_{\text{exact}}(\gamma') && \text{def } [\![-]\!]_{\text{exact}}, \mathcal{T}\ \{-\} \\
&= \big(\lambda x.\textbf{filter}\ (\lambda r.[\![e]\!](\gamma, \delta[\text{ID} \to r]), x)\big)_*[\![t_1]\!](\gamma, \delta) && \text{I.H.} \\
&= [\![t_1\ \textbf{WHERE}\ e]\!](\gamma, \delta) && \text{def } [\![-]\!]
\end{aligned}
$$

- $t \equiv$ **SELECT** e **FROM** t_1

$$
\begin{aligned}
&[\![\mathcal{T}_{\delta'}\ \{\ \textbf{SELECT}\ e\ \textbf{FROM}\ t_1\}]\!]_{\text{exact}}(\gamma') \\
&= [\![\textbf{map}\ (\lambda r.\mathcal{T}_{\delta'[\text{ID}\mapsto r]}\ \{\overline{e}\}, \mathcal{T}_{\delta'}\ \{t\})]\!]_{\text{exact}}(\gamma') && \text{def } \mathcal{T}\ \{\} \\
&= [\![\mathcal{T}_{\delta}\ \{t_1\}]\!]_{\text{exact}}(\gamma') \gg\!= && \text{def } [\![\textbf{map}\ ]\!]_{\text{exact}} \\
&\quad (\lambda S.\textbf{return}\ \{\lambda x'.[\![\mathcal{T}_{\delta'}\ \{e\}]\!]_{\text{exact}}(\gamma'[x \mapsto x'])\ y \mid y \in S\}) \\
&= [\![t_1]\!](\gamma, \delta) \gg\!= (\lambda S.\textbf{return}\ \{\lambda x'.[\![e]\!](\gamma, \delta[x \mapsto x'])\ y \mid y \in S\}) && \text{I.H.} \\
&= [\![\ \textbf{SELECT}\ e\ \textbf{FROM}\ t_1]\!](\gamma, \delta) && \text{def } [\![\ \textbf{SELECT}\ ]\!]
\end{aligned}
$$

- $t \equiv t_1$ **GENERATIVE JOIN** m is similar to the **GENERATE UNDER** case, and is omitted for brevity.
- $t \equiv \text{ID} : T[\text{ID}]\{\text{COLS}\}$ is immediate from the definition of γ and of the ID rules in GenSQL and the lowered language.
- $t \equiv$ ID **GIVEN** c^0 **GIVEN** c^1. y induction hypothesis, we have that $[\![c^0]\!](\gamma, \delta) = [\![\mathcal{T}_{\delta'}\ \{c^0\}]\!]_{\text{exact}}(\gamma')$, and similarly for c^1. Then, by inspection of the semantics of GenSQL and the lowered language, we conclude that the semantics are the same.
- $t \equiv$ **PROBABILITY OF** c^i_1 **UNDER** ID **GIVEN** c^0 **GIVEN** c^1. By induction hypothesis, we have that $[\![c^i_1]\!](\gamma, \delta) = [\![\mathcal{T}_{\delta'}\ \{c^i_1\}]\!]_{\text{exact}}(\gamma')$, and similarly for c^0 and c^1. Then, by inspection of the semantics of GenSQL and the lowered language, we conclude that the semantics are the same.
- $t \equiv t_1 \wedge t_2, t_1 \vee t_2, \text{ID.COL} = e, \text{ID.COL}\ op\ e, op(t_1, \ldots, t_n)$, **RENAME** m **AS** ID$'$, **RENAME** t **AS** ID$'$. These cases are straightforward and immediately follow from the induction hypothesis.
- $t \equiv t_1$ **JOIN** t_2. This case also simply follow from the induction hypothesis, as the **join** operator mimics the behavior of the **JOIN** operator.

D.4 Guarantee for approximate backend

In this section we describe assumptions on approximate rowModels and the approximate semantics of the lowered language. We then state and prove a soundness guarantee for the lowering transform with respect to the approximate semantics. As discussed in Section 4.4, queries with **WHERE** clauses that use approximate values do not necessarily converge to the correct limit even as the approximations converge to the correct value. In this section, we present modifications to the GenSQL and lowered language semantics that detect the types of queries for which this type of behavior can happen. We then give a proof by logical relations for a guarantee that precisely captures the notion of correctness in this setting.

Key insights in the proof. The proof of the guarantee for the approximate backend is based on the following key points:

(1) A safe query (**safe**? macro defined in Fig. 22 returns **true**) is one for which the approximate semantics of the lowered language converges to the exact semantics almost surely.
(2) A query will be safe if its approximations are used in a continuous way. The key theorem is the continuous mapping theorem for random variables which states that if a sequence of

random variables x_n converges to x, then for every continuous function f, $f(x_n)$ converges to $f(x)$.

(3) To ensure this, we define macros **safe**?, **continuous**?, **exact**? (Fig. 22) that track sufficient restrictions on queries to ensure the above. We also define appropriate topologies and metrics on the various spaces of random variables to ensure that the continuous mapping theorem holds.

Topological preliminaries. To apply the continuous mapping theorem, we need to define topologies on the different spaces on which the random variables take values. For the discrete spaces $\mathbb{B}$, $\mathbb{Z}$, $\mathbb{N}$, and **Str**, we use the discrete topology. For the real spaces $\mathbb{R}$ and $\mathbb{R}^+$, we use the usual metric topology. As the spaces also interpret **Null**, we use the discrete topology on **Null**, and the coproduct topology for interpreting base types. For the space of bags, we use the topology of symmetric products [37]. For the product spaces, we use the product topology. All these spaces are metrizable. This is known to be sufficient for the continuous mapping theorem to hold [7][Chapter 5, Section 26]. More precisely, we take the usual metric on base types, extend each of these metrics d to **Null** by defining $d(\textbf{Null}, \textbf{Null}) = 0$, and $d(\textbf{Null}, x) = 1$ for $x \neq \textbf{Null}$. We then take the product metric on product spaces, and the metric on bags as the symmetric product metric. The space of bags with topology of symmetric products on a metric space is known to be metrizable [37]. With these topologies, scalar functions on continuous types are continuous if they're continuous in the usual way. All operations on discrete types are continuous. The extended scalar operations op_s are continuous if the corresponding scalar operation op is continuous (suffices to check that the preimage of the open $\{\textbf{Null}\}$ is the open $\{\textbf{Null}\}$). Likewise, the extended scalar operations op_l are continuous if the corresponding scalar operation op is continuous (suffices to check that preimage of the open $\{\textbf{Null}\}$ is the open $\{\}$). Projections on the i-th component $\textsc{id.col}_i$ are continuous. It remains to show that the operations on bags are continuous. Using the universal property of the bag construction as a colimit in the category of topological spaces and continuous maps, we can show that the bag construction is an endofunctor on the category of topological spaces and continuous maps. This directly implies that **map** is continuous. **singleton** is one of the cocone injection and is therefore continuous by construction. **mapreduce** is the composition of a map and a reduce operation, and is therefore continuous if union is continuous. Likewise, for every n, **replicate** n is simply an n-fold union of the input and is continuous if union is continuous. To show that union is continuous, first note that bag is the countable coproduct of the bags of size n for all n. As finite products distribute over coproducts in the category of topological spaces and continuous maps, we will have a map $\mathbf{Bag}(X) \times \mathbf{Bag}(X) \to \mathbf{Bag}(X)$ if we have a map $\mathbf{Bag}_n(X) \times \mathbf{Bag}_m(X) \to \mathbf{Bag}_{n+m}(X)$ by the universal property of colimits, and where $\mathbf{Bag}_n(X)$ are bags of size n of elements of X. In addition, there's a natural isomorphism between $\mathbf{Bag}_n(X) \times \mathbf{Bag}_m(X)$ and $\mathbf{Bag}_{n+m}(X)$, coming from the isomorphism $X^n \times X^m \simeq X^{n+m}$, which therefore extends to a map from $\mathbf{Bag}_n(X) \times \mathbf{Bag}_m(X) \to \mathbf{Bag}X$, and we are done.

The map **splat** defined by

$$\begin{aligned} \textbf{splat} : (X_1 \times \ldots \times X_n) \times (Y_1 \times \ldots \times Y_m) &\to (X_1 \times \ldots \times X_n \times Y_1 \times \ldots \times Y_m) \\ (x_1, \ldots, x_n), (y_1, \ldots, y_m) &\mapsto (x_1, \ldots, x_n, y_1, \ldots, y_m) \end{aligned}$$

is used to give the semantics of join. This map is a composition of the product of the identity maps $\mathrm{id}_{X_1 \times \cdots \times X_n}$ and $\mathrm{id}_{Y_1 \times \cdots \times Y_m}$ and the isomorphism

$$(X_1 \times \cdots \times X_n) \times (Y_1 \times \cdots \times Y_m) \simeq X_1 \times \cdots \times X_n \times Y_1 \times \cdots \times Y_m.$$

Hence, it is continuous. More generally, given natural numbers n, m, there is a function $X^n \times Y^m \to (X \times Y)^{n \times m}$ which returns all possible combinations of the elements of X and Y in the $n \times m$-tuple. This function is continuous as a composition of identity, duplication, and permutation

(a) **Safeness of Scalar Expressions**

$$\begin{aligned}
\textbf{exact?}(c) = \textbf{continuous?}(c) = \textbf{safe?}(c) &= \textbf{true}\\
\textbf{exact?}(op(e_1,\ldots,e_n)) &= \textbf{exact?}(e_1) \wedge \ldots \wedge \textbf{exact?}(e_n)\\
\textbf{safe?}(op(e_1,\ldots,e_n)) &= \textbf{safe?}(e_1) \wedge \ldots \wedge \textbf{safe?}(e_n)\\
\textbf{continuous?}(op(e_1,\ldots,e_n)) &= \textbf{continuous?}(op) \wedge \bigwedge_{1\le i\le n} \textbf{continuous?}(e_i)\\
\textbf{exact?}(\textsc{id.col}) &= \textbf{false}\\
\textbf{continuous?}(\textsc{id.col}) = \textbf{safe?}(\textsc{id.col}) &= \textbf{true}\\
\textbf{exact?}(\textbf{ PROBABILITY OF } c \textbf{ UNDER } \textsc{id} \textbf{ GIVEN } c^0 \textbf{ GIVEN } c^1) &= \textbf{false}\\
\textbf{safe?}(\textbf{ PROBABILITY OF } c \textbf{ UNDER } \textsc{id} \textbf{ GIVEN } c^0 \textbf{ GIVEN } c^1) &= \textbf{safe?}(c) \wedge \textbf{safe?}(c^0) \wedge \textbf{safe?}(c^1)\\
\textbf{continuous?}(\textbf{ PROBABILITY OF } c \textbf{ UNDER } \textsc{id} \textbf{ GIVEN } c^0 \textbf{ GIVEN } c^1) &= \textbf{safe?}(c) \wedge \textbf{safe?}(c^0) \wedge \textbf{safe?}(c^1)
\end{aligned}$$

(b) **Safeness of Event and Event-0 Expressions**

$$\begin{aligned}
\textbf{safe?}(\textsc{id.col}_i \; op \; e) &= \textbf{exact?}(e)\\
\textbf{safe?}(c \; op \; c') &= \textbf{safe?}(c) \wedge \textbf{safe?}(c')
\end{aligned}$$

(c) **Safeness of Table Expressions**

$$\begin{aligned}
\textbf{safe?}(\textsc{id}) &= \textbf{true}\\
\textbf{safe?}(\textbf{ RENAME } t \textbf{ AS } \textsc{id}) &= \textbf{safe?}(t)\\
\textbf{safe?}(t_1 \textbf{ JOIN } t_2) &= \textbf{safe?}(t_1) \wedge \textbf{safe?}(t_2)\\
\textbf{safe?}(t \textbf{ GENERATIVE JOIN } \textsc{id} \textbf{ GIVEN } c^0 \textbf{ GIVEN } c^1) &= \textbf{safe?}(t) \wedge \textbf{safe?}(c^0) \wedge \textbf{safe?}(c^1)\\
\textbf{safe?}(t \textbf{ WHERE } e) &= \textbf{false}\\
\textbf{safe?}(\textbf{ SELECT } \overline{e} \textbf{ AS } \overline{\textsc{col}} \textbf{ FROM } t) &= \textbf{safe?}(t) \wedge \textbf{continuous?}(e_1) \wedge \ldots \wedge \textbf{continuous?}(e_n)
\end{aligned}$$

Fig. 22. Macros for detecting safe terms for the approximate AMI guarantee

maps. Post-composing with the injection $(X \times Y)^{n\times m} \to \mathbf{Bag}(X \times Y)$, we get a continuous map $X^n \times Y^m \to \mathbf{Bag}(X \times Y)$ which respects the symmetric quotient and therefore extends to a map $\mathbf{Bag}_n(X) \times \mathbf{Bag}_m(Y) \to \mathbf{Bag}(X \times Y)$. We conclude as in the case for union that this extends to the continuous map **join** : $\mathbf{Bag}(X) \times \mathbf{Bag}(Y) \to \mathbf{Bag}(X \times Y)$.

Static analysis for detecting safe terms Fig. 22. The **safe**? macro detects safe terms for which the correctness guarantee for the approximate AMI holds. The **exact**? macro detects those scalar terms whose value does not depend on the approximation used to implement the AMI method. As we formalize below, for such terms the exact and approximate semantics should be almost surely the same. The **continuous**? macro detects those scalar terms that are continuous functions of the results of **prob** or **logpdf** methods.

Random variable semantics. The random variable semantics for the approximate case is given in Fig. 23. If $(\mathcal{X}, \Sigma)$ is a measurable space, then by $\mathrm{StochSeq}(\mathcal{X})$ we denote the set of sequences of $\mathcal{X}$-valued random variables.

For the case of the AMI methods, we assume that the sequences of random variables denoting each method converge to the correct value in the following sense.

Definition D.2 (Asymptotically Sound Approximate AMI Implementation). We say that an implementation of the AMI is asymptotically sound, if for all environments Γ, terms $\Gamma \vdash c^i_j$:

(a) **Semantics of Types and Contexts**

$$\llbracket \Gamma, x : \tau \rrbracket_{\text{approx}} = \llbracket \Gamma \rrbracket_{\text{approx}} \times \llbracket \tau \rrbracket_{\text{approx}} \qquad \llbracket \sigma_g \rrbracket_{\text{approx}} = \text{StochSeq}(\llbracket \sigma_g \rrbracket)$$

$$\llbracket \mathbf{Bag}[\sigma_g] \rrbracket_{\text{approx}} = \text{StochSeq}(\mathcal{P}\mathbf{Bag}\ \llbracket \sigma_g \rrbracket) \qquad \llbracket M[\sigma_g] \rrbracket_{\text{approx}} = \mathcal{P}^{\text{approx}}_{\text{adm}} \llbracket \sigma_g \rrbracket$$

(b) **Semantics of Terms**

$$\llbracket (t_1, \dots, t_n) \rrbracket_{\text{approx}}(\gamma, \omega)_n = (\llbracket t_1 \rrbracket_{\text{approx}}(\gamma, \omega)_n, \dots, \llbracket t_n \rrbracket_{\text{approx}}(\gamma, \omega)_n) \qquad \llbracket c \rrbracket_{\text{approx}}(\gamma, \omega)_n = c$$

$$\llbracket \pi_i\ t \rrbracket_{\text{approx}}(\gamma, \omega)_n = \pi_i(\llbracket t \rrbracket_{\text{approx}}(\gamma, \omega)_n) \qquad \llbracket x \rrbracket_{\text{approx}}(\gamma, \omega)_n = \gamma(x)(\omega)_n$$

$$\llbracket op(t_1, t_2) \rrbracket_{\text{approx}}(\gamma, \omega)_n = op(\llbracket t_1 \rrbracket_{\text{approx}}(\gamma, \omega)_n, \llbracket t_2 \rrbracket_{\text{approx}}(\gamma, \omega)_n)$$

$$\llbracket \mathbf{mapreduce}\ (x.t_1)\ t_2 \rrbracket_{\text{approx}}(\gamma, \omega)_n = l\left[\llbracket t_2 \rrbracket_{\text{approx}}(\gamma, \omega)_n \gg\!= (\lambda S.\ \{\lambda x'.\llbracket t_1 \rrbracket_{\text{approx}}(\gamma[x \mapsto x'], \omega)_n\ y \mid y \in S\})\right]$$

$$\llbracket \mathbf{map}\ (x.t_1)\ t_2 \rrbracket_{\text{approx}}(\gamma, \omega)_n = \llbracket t_2 \rrbracket_{\text{approx}}(\gamma, \omega)_n \gg\!= (\lambda S.\mathbf{return}\ \{\lambda x'.\llbracket t_1 \rrbracket_{\text{approx}}(\gamma[x \mapsto x'], \omega)_n\ y \mid y \in S\})$$

$$\llbracket \mathbf{filter}\ (x.t_1)\ t_2 \rrbracket_{\text{approx}}(\gamma, \omega)_n = \llbracket t_2 \rrbracket_{\text{approx}}(\gamma, \omega)_n \gg\!= (\lambda S.\mathbf{return}\ \{y \in S \mid \lambda x'.\llbracket t_1 \rrbracket_{\text{approx}}(\gamma[x \mapsto x'], \omega)_n\ y\})$$

$$\llbracket \mathbf{replicate}\ (t_1, t_2) \rrbracket_{\text{approx}}(\gamma, \omega)_n = \mathbf{let}\ m = \llbracket t_1 \rrbracket_{\text{approx}}(\gamma, \omega)_n\ \mathbf{in}$$
$$\llbracket t_2 \rrbracket_{\text{approx}}(\gamma, \omega)_n \gg\!= \lambda\mu. \bigotimes_{i=1}^{m} \mu \gg\!= (\lambda(A_1, \dots, A_m).\mathbf{return}\ \cup_{i=1}^{m} A_i)$$

$$\llbracket \mathbf{exp}(t_1) \rrbracket_{\text{approx}}(\gamma, \omega)_n = \exp\ (\llbracket t_1 \rrbracket_{\text{approx}}(\gamma, \omega)_n)$$

$$\llbracket \mathbf{singleton}(t_1) \rrbracket_{\text{approx}}(\gamma, \omega)_n = \mathbf{return}\ \{\llbracket t_1 \rrbracket_{\text{approx}}(\gamma, \omega)_n\}$$

$$\llbracket \mathbf{join}(t_1, t_2) \rrbracket_{\text{approx}}(\gamma, \omega)_n = \llbracket t_1 \rrbracket_{\text{approx}}(\gamma, \omega)_n \otimes \llbracket t_2 \rrbracket_{\text{approx}}(\gamma, \omega)_n \gg\!= (\lambda S, S'.\mathbf{return}\ (\mathbf{map2}\ \mathbf{splat}\ S\ S'))$$

$$\llbracket (\text{ID}, i)\ op\ t : C^1[\sigma_g] \rrbracket_{\text{approx}}(\gamma, \omega)_n = \{(x_1, \dots, x_n) \in \llbracket \sigma_g \rrbracket \mid x_i\ op_l\ \llbracket t \rrbracket_{\text{approx}}(\gamma, \omega)_n\}$$

$$\llbracket (\text{ID}, i) = t : C^0[\sigma_g] \rrbracket_{\text{approx}}(\gamma, \omega)_n = (\pi_i, \llbracket t \rrbracket_{\text{approx}}(\gamma, \omega)_n)$$

$$\llbracket c_1^1 \wedge c_2^1 \rrbracket_{\text{approx}}(\gamma, \omega)_n = \llbracket c_1^1 \rrbracket_{\text{approx}}(\gamma, \omega)_n \cap \llbracket c_2^1 \rrbracket_{\text{approx}}(\gamma, \omega)_n$$

$$\llbracket c_1^1 \vee c_2^1 \rrbracket_{\text{approx}}(\gamma, \omega)_n = \llbracket c_1^1 \rrbracket_{\text{approx}}(\gamma, \omega)_n \cup \llbracket c_2^1 \rrbracket_{\text{approx}}(\gamma, \omega)_n$$

$$\llbracket c_1^0 \wedge c_2^0 \rrbracket_{\text{approx}}(\gamma, \omega)_n = \mathbf{let}\ _{1 \le i \le 2}(f_i, v_i) = \llbracket c_i^0 \rrbracket_{\text{approx}}(\gamma, \omega)_n\ \mathbf{in}\ (\lambda x.(f_1(x), f_2(x)), (v_1, v_2))$$

$$\llbracket \mathbf{simulate}_{\text{ID}}(c^0, c^1) \rrbracket_{\text{approx}}(\gamma, \omega) = \left\{\mu^n_{\text{ID};\llbracket c^0 \rrbracket_{\text{approx}}(\gamma)_n, \llbracket c^1 \rrbracket_{\text{approx}}(\gamma)_n}(\omega)\right\}$$

$$\llbracket \mathbf{logpdf}_{\text{ID}}(c^0, c^1, c_2^0) \rrbracket_{\text{approx}}(\gamma, \omega) = \left\{L^n_{\text{ID};\llbracket c^0 \rrbracket_{\text{approx}}(\gamma)_n, \llbracket c^1 \rrbracket_{\text{approx}}(\gamma)_n, \llbracket c_2^0 \rrbracket_{\text{approx}}(\gamma)_n}(\omega)\right\}$$

$$\llbracket \mathbf{prob}_{\text{ID}}(c_2^1, c^0, c^1) \rrbracket_{\text{approx}}(\gamma, \omega) = \left\{P^n_{\text{ID};\llbracket c^0 \rrbracket_{\text{approx}}(\gamma)_n, \llbracket c^1 \rrbracket_{\text{approx}}(\gamma)_n, \llbracket c_2^1 \rrbracket_{\text{approx}}(\gamma)_n}(\omega)\right\}$$

Fig. 23. Approximate semantics of the lowered language.

$C^i[(\sigma_1, \dots, \sigma_n)]$, and all $\gamma \in \llbracket \Gamma \rrbracket$, we have

$$\llbracket \mathbf{logpdf}_{\text{ID}}(c_1^0, c_1^1, c_2^0) \rrbracket_{\text{approx}}(\gamma)_n \longrightarrow \llbracket \mathbf{logpdf}_{\text{ID}}(c_1^0, c_1^1, c_2^0) \rrbracket_{\text{exact}}(\gamma)$$
$$\llbracket \mathbf{prob}_{\text{ID}}(c_1^0, c_1^1, c_2^1) \rrbracket_{\text{approx}}(\gamma)_n \longrightarrow \llbracket \mathbf{prob}_{\text{ID}}(c_1^0, c_1^1, c_2^1) \rrbracket_{\text{exact}}(\gamma)$$
$$\llbracket \mathbf{simulate}_{\text{ID}}(c_1^0, c_1^1) \rrbracket_{\text{approx}}(\gamma)_n \longrightarrow \llbracket \mathbf{simulate}_{\text{ID}}(c_1^0, c_1^1) \rrbracket_{\text{exact}}(\gamma),$$

$\mathbb{P}$-almost surely.

The implementation of such asymptotically sound estimators in presence of nested conditioning is considered in the Bayesian inference literature [47, 62]. We denote by $\mathcal{P}^{\text{approx}}_{\text{adm}} X$ the set of asymptotically sound approximate AMI implementation where ID is a model on X. Given an evaluation context γ, we denote by γ_n the following mapping. If $\gamma = \{\text{ID}_1 \mapsto a^1, \dots, \text{ID}_m \mapsto a^m\}$, then $\gamma_n := \{\text{ID}_1 \mapsto a_n^1, \dots, \text{ID}_m \mapsto a_n^m \dots\}$, where a_n^i is a^i if ID_i is a table type, and the n-th element of the sequence a^i if ID_i is a model type. We can then prove the following lemma by simple induction on the structure of the term.

Logical Relations

$$\begin{aligned}
(x, \{X_n\}) \in R_\sigma &\iff {}_n \to x \text{ a.s.} \\
((f, v), (g, \{X_n\})) \in R_{C^0[\text{COLS}]} &\iff f = g \wedge (v = X_n \text{ almost surely for all } n \in \mathbb{N}) \\
(E, \{E_n\}) \in R_{C^1[\text{COLS}]} &\iff E = E_n \text{ almost surely for all } n \in \mathbb{N} \\
(\mu, \{\mu_n\}) \in R_{\mathcal{T}} &\iff \mu_n(\omega) \Rightarrow \mu \text{ for } \mathbb{P}\text{-almost all } \omega
\end{aligned}$$

Fig. 24. Logical relations used for the proof of soundness of $\mathcal{T}\{\cdot\}$ with respect to the approximate semantics.

Lemma D.3. *If a term $\Gamma; \Delta \vdash t : \tau$ is exact then $\mathcal{P}$-almost surely $[\![\mathcal{T}_\delta\{t\}]\!]_{approx}(\gamma)_n = [\![\mathcal{T}_\delta\{t\}]\!]_{exact}(\gamma_n)$ for all n and all evaluation contexts γ, δ.*

The proof of soundness of the translation $\mathcal{T}\{\cdot\}$ relies on logical relations given in Fig. 24. The logical relations are in terms of convergence of sequences taking values in $[\![\tau]\!]$ for different types τ. We assume $\mathbb{B}$, $\mathbb{Z}$, $\mathbb{N}$, and **Str** have the discrete topology and $\mathbb{R}$ and $\mathbb{R}^+$ have their usual metric topology. Product spaces are endowed with the usual product topology and the space of bags is endowed with the topology of symmetric products [37]. The fundamental lemma of logical relations, which implies the soundness of our translation, is as follows.

Lemma D.4 (Fundamental lemma of logical relations). *Suppose*

$$\Gamma := x_1 : \tau_1, \ldots, x_n : \tau_n \qquad \Delta := x'_1 : \tau'_1, \ldots, x'_m : \tau'_m$$

and $\Gamma; \Delta \vdash t : \tau$ where t is a normalized GenSQL term which is safe (continuous for a scalar type, safe for a table, event or event-0 type). Take any $a_i \in [\![\tau_i]\!]$, $a'_i \in [\![\tau'_i]\!]$, $b_i \in [\![\mathcal{T}\{\tau_i\}]\!]_{approx}$, and $b'_i \in [\![\mathcal{T}\{\tau'_i\}]\!]_{approx}$, such that $(a_i, b_i) \in R_{\tau_i}$ and $(a'_i, b'_i) \in R_{\tau'_i}$. Moreover, suppose that the implementation of the AMI methods are asymptotically sound. Then, for all $\delta_{\mathcal{T}}$ of the form

$$\delta_{\mathcal{T}} = \{x'_i \mapsto r_i | 1 \leq i \leq m\}$$

where the r_is are variable names in the lowered language, if we let

$$\gamma := \{x_i \mapsto a_i | 1 \leq i \leq n\} \qquad \delta := \{x'_i \mapsto a'_i | 1 \leq i \leq m\} \qquad \delta' := \{r_i \mapsto b'_i | 1 \leq i \leq m\}$$

we will have

$$([\![t]\!](\gamma, \delta), [\![\mathcal{T}_{\delta_{\mathcal{T}}}\{t\}]\!]_{approx}(\mathcal{T}\{\gamma\} \cup \delta')) \in R_\tau.$$

proof sketch. We prove the lemma case by case for normalized GenSQL queries.

Scalar Expressions

constants If t is a constant of type σ, then then $\mathcal{T}_{\delta_{\mathcal{T}}}\{t\} = t$ by the lowering rules and so

$$[\![t]\!](\gamma, \delta) = t = [\![t]\!]_{\text{approx}}(\mathcal{T}\{\gamma\}) = [\![\mathcal{T}_{\delta_{\mathcal{T}}}\{t\}]\!]_{\text{approx}}.$$

As constant sequences are almost surely convergent to their value, we can conclude that $([\![t]\!](\gamma, \delta), [\![\mathcal{T}_{\delta_{\mathcal{T}}}\{t\}]\!]_{\text{approx}}(\mathcal{T}\{\gamma\} \cup \delta')) \in R_\sigma$.

column values If t is of the form ID.COL, then by the typing rules of GenSQL the local context Δ should contain $T[\text{ID}]\{\text{COLS}\}$ as its last entry. Hence, either by our assumption on the context, or by the induction hypothesis, ID is a safe term. As projection is a continuous operation, we have that

$$\begin{aligned}&[\![\mathcal{T}_{\delta_{\mathcal{T}}}\{\textsc{id.col}_i\}]\!]_{\text{approx}}(\mathcal{T}\{\gamma\})_n\\ &= \pi_i([\![\mathcal{T}_{\delta_{\mathcal{T}}}\{\textsc{id}\}]\!]_{\text{approx}}(\mathcal{T}\{\gamma\})(\mathcal{T}\{\gamma\}))_n\\ &\xrightarrow{\text{a.s.}} \pi_i([\![\textsc{id}]\!](\gamma,\delta))\\ &= [\![\textsc{id.col}_i]\!](\gamma,\delta)\end{aligned}$$

mathematical operations If t is of the form $op(e_1,\dots,e_n)$ where $\Gamma;\Delta\vdash e_i:\sigma_i$ and **continuous**?(op), by definition of R_{σ_i} and our assumption we must have $[\![\mathcal{T}_{\delta_{\mathcal{T}}}\{e_i\}]\!]_{\text{approx}}(\mathcal{T}\{\gamma\})_n \xrightarrow{\text{a.s.}} [\![e_i]\!]$. By the lowering rules we have

$$[\![\mathcal{T}_{\delta_{\mathcal{T}}}\{op(e_1,\dots,e_n)\}]\!]_{\text{approx}}(\mathcal{T}\{\gamma\}) = op([\![\mathcal{T}_{\delta_{\mathcal{T}}}\{e_1\}]\!]_{\text{approx}}(\mathcal{T}\{\gamma\}),\dots,[\![\mathcal{T}_{\delta_{\mathcal{T}}}\{e_n\}]\!]_{\text{approx}}(\mathcal{T}\{\gamma\})),$$

As op is continuous, we can conclude that

$$[\![\mathcal{T}_{\delta_{\mathcal{T}}}\{op(e_1,\dots,e_n)\}]\!]_{\text{approx}}(\mathcal{T}\{\gamma\})_n \xrightarrow{\text{a.s.}} [\![op(e_1,\dots,e_n)]\!](\gamma,\delta).$$

Hence, by definition of R_σ our claim holds.

probability of an event If t is of the form PROBABILITY OF c UNDER m with $\Gamma,\Delta\vdash c:C^1\{\textsc{cols}\}$, then by normalization assumption t must necessarily be in the normal form

$$\textbf{PROBABILITY OF } c \textbf{ UNDER } \textsc{id} \textbf{ GIVEN } c^0 \textbf{ GIVEN } c^1.$$

Now,

$$\begin{aligned}&[\![\mathcal{T}_{\delta_{\mathcal{T}}}\{\textbf{ PROBABILITY OF } c \textbf{ UNDER } \textsc{id} \textbf{ GIVEN } c^0 \textbf{ GIVEN } c^1\}]\!]_{\text{approx}}(\mathcal{T}\{\gamma\})_n\\ &\quad= [\![\textbf{prob}\mathcal{T}_{\delta_{\mathcal{T}}}\{c^0\}\,\mathcal{T}_{\delta_{\mathcal{T}}}\{c^1\}\,\mathcal{T}_{\delta_{\mathcal{T}}}\{c\}]\!]_{\text{approx}}(\mathcal{T}\{\gamma\})_n && \text{(lowering rules)}\\ &\quad\xrightarrow{\text{a.s.}} [\![\textbf{prob}\mathcal{T}_{\delta_{\mathcal{T}}}\{c^0\}\,\mathcal{T}_{\delta_{\mathcal{T}}}\{c^1\}\,\mathcal{T}_{\delta_{\mathcal{T}}}\{c\}]\!]_{\text{exact}}(\mathcal{T}\{\gamma\}) && \text{(sound AMI implementation)}\\ &\quad= [\![\mathcal{T}_{\delta_{\mathcal{T}}}\{\textbf{ PROBABILITY OF } c \textbf{ UNDER } \textsc{id} \textbf{ GIVEN } c^0 \textbf{ GIVEN } c^1\}]\!]_{\text{exact}}(\mathcal{T}\{\gamma\}) && \text{(lowering rules)}\\ &\quad= [\![\textbf{ PROBABILITY OF } c \textbf{ UNDER } \textsc{id} \textbf{ GIVEN } c^0 \textbf{ GIVEN } c^1]\!](\gamma). && \text{(exact guarantee)}\end{aligned}$$

This is a base case for the induction.

likelihood of an event-0 If t is of the form PROBABILITY OF c UNDER m with $\Gamma,\Delta\vdash c:C^0\{\textsc{cols}'\}$, then the proof is almost identical to the case where $\Gamma,\Delta\vdash c:C^1\{\textsc{cols}\}$.

Event Expressions

primitive events Suppose $\Gamma;\Delta\vdash t:C^1\{\textsc{cols}\}$ is of the form $\textsc{id.col}_i\ op\ e$ where

$$\textsc{cols} = \textsc{col}_1:\sigma_1,\dots,\textsc{col}_m:\sigma_m.$$

For ease of notation we let $\sigma_g := (\sigma_1,\dots,\sigma_m)$. By our safety assumption and the definition of the **safe**? macro, the subterm e must be exact. By Lemma D.3, we must then have $[\![\mathcal{T}_{\delta_{\mathcal{T}}}\{e\}]\!]_{\text{approx}}(\mathcal{T}\{\gamma\})_n = [\![\mathcal{T}_{\delta_{\mathcal{T}}}\{e\}]\!]_{\text{exact}}(\mathcal{T}\{\gamma\})$ for all n almost surely. Hence, by definition of the lowering rules and the approximate semantics

$$\begin{aligned}[\![\mathcal{T}_{\delta_{\mathcal{T}}}\{\textsc{id.col}_i\ op\ e\}]\!]_{\text{approx}}(\mathcal{T}\{\gamma\})_n &= [\![(\textsc{id},i)\ op\ \mathcal{T}_{\delta_{\mathcal{T}}}\{e\}]\!]_{\text{approx}}(\mathcal{T}\{\gamma\})_n\\ &= \{(x_1,\dots,x_m)\in[\![\sigma_g]\!] \mid x_i\ op\ [\![\mathcal{T}_{\delta_{\mathcal{T}}}\{e\}]\!]_{\text{approx}}(\mathcal{T}\{\gamma\})_n\}\\ &= \{(x_1,\dots,x_m)\in[\![\sigma_g]\!] \mid x_i\ op\ [\![\mathcal{T}_{\delta_{\mathcal{T}}}\{e\}]\!]_{\text{exact}}(\mathcal{T}\{\gamma\})\}\\ &= [\![(\textsc{id},i)\ op\ \mathcal{T}_{\delta_{\mathcal{T}}}\{e\}]\!]_{\text{exact}}(\mathcal{T}\{\gamma\})\\ &= [\![\mathcal{T}_{\delta_{\mathcal{T}}}\{\textsc{id.col}_i\ op\ e\}]\!]_{\text{exact}}(\mathcal{T}\{\gamma\}),\end{aligned}$$

almost surely. Applying the soundness theorem for the exact AMI implementations, we are done.

conjunctions and disjunctions If $\Gamma; \Delta \vdash t : C^1\{\textsc{cols}\}$ is of the form $c^1_1 \wedge c^1_2$, then by our inductive assumption we must have

$$[\![c^1_i]\!](\gamma, \delta) = [\![\mathcal{T}_{\delta_\mathcal{T}}\left\{c^1_i\right\}]\!]_{\text{approx}}(\mathcal{T}\{\gamma\})_n$$

almost surely for all n. Hence,

$$\begin{aligned}[\![\mathcal{T}_{\delta_\mathcal{T}}\left\{c^1_1 \wedge c^2_2\right\}]\!]_{\text{approx}}(\mathcal{T}\{\gamma\})_n &= [\![\mathcal{T}_{\delta_\mathcal{T}}\left\{c^1_1\right\} \wedge \mathcal{T}_{\delta_\mathcal{T}}\left\{c^1_2\right\}]\!]_{\text{approx}}(\mathcal{T}\{\gamma\})_n \\ &= [\![\mathcal{T}_{\delta_\mathcal{T}}\left\{c^1_1\right\}]\!]_{\text{approx}}(\mathcal{T}\{\gamma\})_n \cap [\![\mathcal{T}_{\delta_\mathcal{T}}\left\{c^1_2\right\}]\!]_{\text{approx}}(\mathcal{T}\{\gamma\})_n \\ &= [\![c^1_1]\!](\gamma, \delta) \cap [\![c^1_2]\!](\gamma, \delta) \\ &= [\![c^1_1 \wedge c^1_2]\!](\gamma, \delta)\end{aligned}$$

The proof is almost identical when $\Gamma; \Delta \vdash t : C^1\{\textsc{cols}\}$ is of the form $c^1_1 \vee c^1_2$.

Event-0 Expressions

primitive event-0s Suppose $\Gamma; \Delta \vdash t : C^0\{\textsc{col}_i\}$ is of the form $\textsc{id}.\textsc{col}_i = e$ where $\textsc{col}_i : \sigma$. By our safety assumption and the definition of the **safe**? macro, the subterm e must be exact. By Lemma D.3, we must then have $[\![\mathcal{T}_{\delta_\mathcal{T}}\{e\}]\!]_{\text{approx}}(\mathcal{T}\{\gamma\})_n = [\![\mathcal{T}_{\delta_\mathcal{T}}\{e\}]\!]_{\text{exact}}(\mathcal{T}\{\gamma\})$ for all n almost surely. Hence, by definition of the lowering rules and the approximate semantics

$$\begin{aligned}[\![\mathcal{T}_{\delta_\mathcal{T}}\{\textsc{id}.\textsc{col}_i = e\}]\!]_{\text{approx}}(\mathcal{T}\{\gamma\})_n &= [\![(\textsc{id}, i) = \mathcal{T}_{\delta_\mathcal{T}}\{e\}]\!]_{\text{approx}}(\mathcal{T}\{\gamma\})_n \\ &= (\pi_i, [\![\mathcal{T}_{\delta_\mathcal{T}}\{e\}]\!]_{\text{approx}}(\mathcal{T}\{\gamma\})_n) \\ &= (\pi_i, [\![\mathcal{T}_{\delta_\mathcal{T}}\{e\}]\!]_{\text{exact}}(\mathcal{T}\{\gamma\})) \\ &= [\![(\textsc{id}, i) = \mathcal{T}_{\delta_\mathcal{T}}\{e\}]\!]_{\text{exact}}(\mathcal{T}\{\gamma\}) \\ &= [\![\mathcal{T}_{\delta_\mathcal{T}}\{\textsc{id}.\textsc{col}_i\ op\ e\}]\!]_{\text{exact}}(\mathcal{T}\{\gamma\}),\end{aligned}$$

almost surely. By the soundness theorem for the exact AMI implementations, we are done.

conjunctions If $\Gamma; \Delta \vdash t : C^0\{\textsc{cols}\}$ is of the form $c^1_0 \wedge c^1_0$, then by our inductive assumption we must have

$$[\![c^0_i]\!](\gamma, \delta) = [\![\mathcal{T}_{\delta_\mathcal{T}}\left\{c^0_i\right\}]\!]_{\text{approx}}(\mathcal{T}\{\gamma\})_n$$

almost surely for all n. Hence,

$$\begin{aligned}[\![\mathcal{T}_{\delta_\mathcal{T}}\left\{c^0_1 \wedge c^0_2\right\}]\!]_{\text{approx}}(\mathcal{T}\{\gamma\})_n &= [\![\mathcal{T}_{\delta_\mathcal{T}}\left\{c^0_1\right\} \wedge \mathcal{T}_{\delta_\mathcal{T}}\left\{c^0_2\right\}]\!]_{\text{approx}}(\mathcal{T}\{\gamma\})_n \\ &= \textbf{let}\ {}_{1\le i\le 2}(f_i, v_i) = [\![\mathcal{T}_{\delta_\mathcal{T}}\left\{c^0_i\right\}]\!]_{\text{approx}}(\mathcal{T}\{\gamma\})_n\ \textbf{in} \\ &\qquad \big(\lambda x.(f_1(x), f_2(x)), (v_1, v_2)\big) \\ &= \textbf{let}\ {}_{1\le i\le 2}(f_i, v_i) = [\![c^0_i]\!](\gamma, \delta)\ \textbf{in} \\ &\qquad \big(\lambda x.(f_1(x), f_2(x)), (v_1, v_2)\big) \\ &= [\![c^0_1 \wedge c^0_2]\!](\gamma, \delta),\end{aligned}$$

almost surely.

Table Expressions

loaded tables If $\Gamma; \Delta \vdash \textsc{id} : T[\textsc{id}]\{\textsc{cols}\}$, then by assumption $\textsc{id}$ is safe and there's nothing to show.

renamed tables If t is of the form **RENAME** t' **AS** $\textsc{id}'$ then by the typing rules of GenSQL we must have $\Gamma; \Delta \vdash t' : T[?\textsc{id}]\{\textsc{cols}\}$, and so by our inductive assumption

$$[\![\mathcal{T}_{\delta_\mathcal{T}}\{t'\}]\!]_{\text{approx}}(\mathcal{T}\{\gamma\})_n \xrightarrow{\text{a.s.}} [\![t']\!](\gamma, \delta).$$

By definition of $\mathcal{T}_{\delta_\mathcal{T}}\{-\}$ we have

$$\begin{aligned}\llbracket \mathcal{T}_{\delta_\mathcal{T}}\{\ \textbf{RENAME}\ t'\ \textbf{AS}\ \textsc{id}'\}\rrbracket_{\text{approx}}(\mathcal{T}\{\gamma\})_n &= \llbracket \mathcal{T}_{\delta_\mathcal{T}}\{t'\}\rrbracket_{\text{approx}}(\mathcal{T}\{\gamma\})_n \\ &\xrightarrow{\text{a.s.}} \llbracket t'\rrbracket(\gamma,\delta) \\ &= \llbracket\ \textbf{RENAME}\ t'\ \textbf{AS}\ \textsc{id}'\rrbracket(\gamma,\delta).\end{aligned}$$

joins If t is of the form t_1 **JOIN** t_2 then by the typing rules of GenSQL we must have $\Gamma;\Delta \vdash t_i : T[?\textsc{id}_i]\{\textsc{cols}_i\}$, and so by our inductive assumption

$$\llbracket \mathcal{T}_{\delta_\mathcal{T}}\{t_i\}\rrbracket_{\text{approx}}(\mathcal{T}\{\gamma\})_n \xrightarrow{\text{a.s.}} \llbracket t\rrbracket(\gamma,\delta).$$

For ease of notation, we let $\sigma_g = (\sigma_1,\dots,\sigma_n)$ and $\sigma'_g = (\sigma'_1,\dots,\sigma'_m)$ where Now,

$$\begin{aligned}\llbracket \mathcal{T}_{\delta_\mathcal{T}}\{t_1\ \textbf{JOIN}\ t_2\}\rrbracket_{\text{approx}}(\mathcal{T}\{\gamma\})_n &= \llbracket \textbf{join}(\mathcal{T}_{\delta_\mathcal{T}}\{t_1\}, \mathcal{T}_{\delta_\mathcal{T}}\{t_2\})\rrbracket_{\text{approx}}(\mathcal{T}\{\gamma\})_n \\ &= \llbracket t_1\rrbracket_{\text{approx}}(\mathcal{T}\{\gamma\})_n \otimes \llbracket t_2\rrbracket_{\text{approx}}(\mathcal{T}\{\gamma\})_n \gg\!= \\ &\quad (\lambda S,S'.\textbf{return}\ (\textbf{map2 splat}\ S\ S'))\end{aligned}$$

As described in our topological preliminaries above, the mappings **splat** , **join** and **map2** are continuous. Using the continuous mapping theorem and the basic properties of weak convergence of measures we can pass the above to the limit and apply our inductive hypothesis to get

$$\begin{aligned}\llbracket \mathcal{T}_{\delta_\mathcal{T}}\{t_1\ \textbf{JOIN}\ t_2\}\rrbracket_{\text{approx}}(\mathcal{T}\{\gamma\})_n &\Rightarrow \llbracket t_1\rrbracket(\gamma,\delta)\otimes\llbracket t_2\rrbracket(\gamma,\delta) \gg\!= \\ &\quad (\lambda S,S'.\textbf{return}\ (\textbf{map2 splat}\ S\ S')) \\ &= \llbracket t_1\ \textbf{JOIN}\ t_2\rrbracket(\gamma,\delta).\end{aligned}$$

filtered tables If t is of the form t **WHERE** e then by definition of the **safe**? macro t has to be unsafe. Hence, there is nothing to show in this case.

generated tables If t is of the form **GENERATE UNDER** m **LIMIT** e, by the normalization assumption t must be of the form **GENERATE UNDER** ID **GIVEN** c^0 **GIVEN** c^1 **LIMIT** e. By the lowering rules,

$$\begin{aligned}&\llbracket \mathcal{T}_{\delta_\mathcal{T}}\left\{\ \textbf{GENERATE UNDER}\ \textsc{id}\ \textbf{GIVEN}\ c^0\ \textbf{GIVEN}\ c^1\ \textbf{LIMIT}\ e\right\}\rrbracket_{\text{approx}}(\mathcal{T}\{\gamma\})_n = \\ &\qquad\qquad \llbracket \textbf{replicate}\ \mathcal{T}_{\delta_\mathcal{T}}\{e\}, \textbf{simulate}_{\textsc{id}}(\mathcal{T}_{\delta_\mathcal{T}}\{c^0\}, \mathcal{T}_{\delta_\mathcal{T}}\{c^1\})\rrbracket_{\text{approx}}(\mathcal{T}\{\gamma\})_n\end{aligned}$$

Now, by definition of **safe**? and **exact**? and Lemma D.3 we must have

$$\begin{aligned}\llbracket \mathcal{T}_{\delta_\mathcal{T}}\{c^0\}\rrbracket_{\text{approx}}(\mathcal{T}\{\gamma\})_n &= \llbracket \mathcal{T}_{\delta_\mathcal{T}}\{c^0\}\rrbracket_{\text{exact}}(\mathcal{T}\{\gamma\}) \\ \llbracket \mathcal{T}_{\delta_\mathcal{T}}\{c^1\}\rrbracket_{\text{approx}}(\mathcal{T}\{\gamma\})_n &= \llbracket \mathcal{T}_{\delta_\mathcal{T}}\{c^1\}\rrbracket_{\text{exact}}(\mathcal{T}\{\gamma\}) \\ \llbracket \mathcal{T}_{\delta_\mathcal{T}}\{e\}\rrbracket_{\text{approx}}(\mathcal{T}\{\gamma\})_n &= \llbracket \mathcal{T}_{\delta_\mathcal{T}}\{e\}\rrbracket_{\text{exact}}(\mathcal{T}\{\gamma\}),\end{aligned}$$

almost surely for all n. Hence, using the definition of the approximate semantics we can see that

$$\begin{aligned}&\llbracket \textbf{replicate}\ \mathcal{T}_{\delta_\mathcal{T}}\{e\}, \textbf{simulate}_{\textsc{id}}(\mathcal{T}_{\delta_\mathcal{T}}\{c^0\}, \mathcal{T}_{\delta_\mathcal{T}}\{c^1\})\rrbracket_{\text{approx}}(\mathcal{T}\{\gamma\}) \\ &= \llbracket \textbf{replicate}\ \mathcal{T}_{\delta_\mathcal{T}}\{e\}, \textbf{simulate}_{\textsc{id}}(\mathcal{T}_{\delta_\mathcal{T}}\{c^0\}, \mathcal{T}_{\delta_\mathcal{T}}\{c^1\})\rrbracket_{\text{exact}}(\mathcal{T}\{\gamma\})_n\end{aligned}$$

almost surely for all n. Applying the correctness result of the exact AMI we are done.

selects By assumption, the term e in the select is continuous, and by assumption the term t is safe. Hence, the continuous mapping theorem directly applies as the semantics of select is the pushforward by the semantics of e of the semantics of t.

generative joins by assumption, $\textbf{simulate}_{\textsc{id}}$ converges to the exact semantics almost surely. Singleton and pairing are continuous, as well as join and mapreduce, and so the whole translation of the generative join is continuous and converges to the exact semantics almost surely.

Row Model Expressions As the lowering is defined on normalized queries, all rowModel terms are of the form ID GIVEN c^0 GIVEN c^1, where the GIVEN clause are optional. Therefore, these terms are never directly returned by the lowering transformation, and are only used in PROBABILITY OF and GENERATE UNDER expressions. Hence, the proof is complete. □

Using the fundamental lemma, we can prove the following result on the soundness of the lowering transformation in presence of approximate semantics.

THEOREM D.5 (CONSISTENT AMI GUARANTEE). *Let* $\Gamma, [] \vdash t : T[?\textsc{id}]\{\textsc{cols}\}$ *be a safe query and suppose the AMI methods have asymptotically sound implementations. Then, for all evaluation context* γ*, we have*

$$\lim_n([\![\mathcal{T}_{[]}\ \{t\}]\!]_{approx}(\mathcal{T}\ \{\gamma\})) = [\![t]\!](\lim_n \gamma)$$

$\mathbb{P}$*-almost surely.*

PROOF. Since the AMI methods are assumed to have asymptotically sound implementations and the queyr safe, by the fundamental lemma we must have, for all γ and $\delta = []$, that

$$([\![t]\!](\gamma, []), [\![\mathcal{T}_{[]}\ \{t\}]\!]_{\text{approx}}(\mathcal{T}\ \{\gamma\}, [])) \in R_{T[?\textsc{id}]\{\textsc{cols}\}}$$

By the second part of the definition of the logical relation for table types, we have that almost surely $\lim_n([\![\mathcal{T}_{[]}\ \{t\}]\!]_{\text{approx}}(\gamma)) = [\![t]\!](\lim_n \gamma)$, as required. □

We finally note that we gave a sufficient but not necessary condition for the soundness of the approximate semantics of the lowered language, and leave the development of a more permissive safe macro for future work.

E FULL LANGUAGE

Our implementation of GenSQL contains several more primitives and operations than the core language described in the main text. Here, we present a fuller language more representative of the implementation. Yet, we still omit several constructs such as SQL's LEFT JOIN, RIGHT JOIN, and OUTER JOIN which can be expressed with the primitives we present. These constructs are omitted for brevity and because they are not essential to the core functionality of the language, but they can of course be convenient and have optimized implementations in practice.

We assume given a set of aggregates $A \in \mathcal{A}$. For aggregates A, we write $A : \sigma_1 \longrightarrow \sigma_2$ to indicate that the aggregate operates on elements on type σ_1 and returns elements of type σ_2. Figure 25 shows a list of common aggregates and their types. DEDUP removes duplicates rows in a table, while DUPLICATE creates copies of each row in a table. WITH t_1 AS ID$'$: t_2 is a convenient notation for introducing a new table name ID$'$ bound to the value of t_1 in the context of a table expression t_2. It is a SQL equivalent of a let binding. It can be used jointly with GENERATIVE JOIN to generate multiple possible completions of a row under a model. MUTUAL INFO computes the conditional mutual information between two sets of columns in a table, conditioned on an event or event-0 c^i under a model m. In general, event if marginal densities can be computed exactly for a model m, the (conditional) mutual information may not be computable in closed form, and is typically approximated using Monte Carlo methods [67]. We can for instance use MUTUAL INFO as a compact way to express the computation presented in Fig. 2.

The example in Fig. 2 uses a syntax closer to SQL in its use of GROUP BY . It is possible to express the same query using the syntax in Fig. 26 by using its GROUP BY operator, as follows. Let t be the query from lines 2-17 in Fig. 2. Then the query in Fig. 2 can be expressed as GROUP t BY [table.weight AS weight] AGGREGATING Avg(table.log_pxy_div_px_py) AS mutual_info.

Aggregate	Types
Sum	$\sigma_c \to$ **Real**, **Int** $\to$ **Int**, **Nat** $\to$ **Nat**
Avg	**Int** $\to$ **Real**, **Nat** $\to$ **Real**, $\sigma_c \to \sigma_c$
Max, Min	$\sigma_c \to \sigma_c$, **Int** $\to$ **Int**, **Nat** $\to$ **Nat**, **Bool** $\to$ **Bool**
Count, CountDistinct	$\sigma \to$ **Nat**
Concat	**Str** $\to$ **Str**

Fig. 25. Supported types of aggregate operators.

$$
\begin{aligned}
\text{Continuous base types } \sigma_c &::= \mathbf{Real} \mid \mathbf{PosReal} \mid \mathbf{Ranged}(a, b) \\
\text{Discrete base types } \sigma_d &::= \mathbf{Int} \mid \mathbf{Str} \mid \mathbf{Nat} \mid \mathbf{Bool} \mid \mathbf{Cat}(\textsc{n}_1, \dots, \textsc{n}_k) \\
\text{Base Types } \sigma &::= \sigma_c \mid \sigma_d \\
\text{Table Types } \mathcal{T} &::= T[?\textsc{id}]\{\textsc{col}_1 : \sigma_1, \dots, \textsc{col}_n : \sigma_n\} \\
\text{rowModel Types } \mathcal{M} &::= M[?\textsc{id}]\{\textsc{col}_1 : \sigma_1, \dots, \textsc{col}_n : \sigma_n\} \\
\text{Event Types } \mathcal{E} &::= C^1\{\textsc{col}_1 : \sigma_1, \dots, \textsc{col}_n : \sigma_n\} \mid C^0\{\textsc{col}_1 : \sigma_1, \dots, \textsc{col}_n : \sigma_n\} \\
\text{Aggregates } A &::= \mathbf{Sum} \mid \mathbf{Avg} \mid \mathbf{Max} \mid \mathbf{Min} \mid \mathbf{Count} \mid \mathbf{Concat} \mid \mathbf{CountDistinct} \\
\text{Operations } op &::= + \mid * \mid \div \mid \log \mid \exp \mid \sqrt{\cdot} \mid > \mid < \mid = \mid \dots \\
\text{Table Expressions } t &::= \textsc{id} \mid t_1 \textbf{ UNION } t_2 \mid t_1 \textbf{ JOIN } t_2 \mid \textbf{RENAME } t \textbf{ AS } \textsc{id} \\
&\quad \mid \textbf{DEDUP } t \mid t \textbf{ DUPLICATE } e \textbf{ TIMES } \mid t \textbf{ WHERE } e \mid \textbf{WITH } t_1 \textbf{ AS } \textsc{id}' : t_2 \\
&\quad \mid \textbf{SELECT } e_1 \textbf{ AS } \textsc{col}_1, \dots, e_n \textbf{ AS } \textsc{col}_n \textbf{ FROM } t \\
&\quad \mid \textbf{GROUP } t \textbf{ BY } [e_1 \textbf{ AS } \textsc{col}_1, \dots, e_n \textbf{ AS } \textsc{col}_n] \\
&\qquad \textbf{AGGREGATING } A_1(e'_1) \textbf{ AS } \textsc{col}'_1, \dots, A_m(e'_m) \textbf{ AS } \textsc{col}'_m \\
&\quad \mid \textbf{GENERATE UNDER } m \textbf{ LIMIT } e \mid t \textbf{ GENERATIVE JOIN } m \textbf{ GIVEN } c^i \\
\text{rowModel Expressions } m &::= \textsc{id} \mid m \textbf{ GIVEN } c^i \mid \textbf{RENAME } m \textbf{ AS } \textsc{id} \\
\text{Scalar Expressions } e &::= \textsc{id.col} \mid op(e_1, \dots, e_n) \mid \textbf{PROBABILITY OF } c^i \textbf{ UNDER } m \\
&\quad \mid \textbf{MUTUAL INFO } (\textsc{id.cols}, \textsc{id.cols}', c^i) \textbf{ UNDER } m \\
\text{Event Expressions } c^1 &::= \bigwedge_{1 \le i \le 2} c^1_i \mid \bigvee_{1 \le i \le 2} c^1_i \mid \textsc{id.col}\ op\ e \\
\text{Event-0 Expressions } c^0 &::= \bigwedge_{1 \le i \le 2} c^0_i \mid \textsc{id.col} = e
\end{aligned}
$$

Fig. 26. Full syntax of GenSQL.

F LIST OF GENSQL QUERIES

Figure 29 showcases the modelling capabilities of GenSQL on a variety of data analysis examples. Figure 30 shows the queries used from the benchmark in Table 1.

Received 2023-11-16; accepted 2024-03-31

$$\frac{\Gamma;\Delta \vdash t : T[?\textsc{id}]\{\textsc{cols}\}}{\Gamma;\Delta \vdash \textsc{dedup}\ t : T[?\textsc{id}]\{\textsc{cols}\}} \qquad \frac{\Gamma;\Delta \vdash t_1 : T[\textsc{id}_1]\{\textsc{cols}\} \quad \Gamma;\Delta \vdash t_2 : T[\textsc{id}_2]\{\textsc{cols}\}}{\Gamma;\Delta \vdash t_1\ \textsc{union}\ t_2 : T[]\{\textsc{cols}\}}$$

$$\frac{\Gamma;\Delta \vdash t : T[?\textsc{id}]\{\textsc{cols}\} \quad \Gamma;\Delta \vdash e : \mathbf{Nat}}{\Gamma;\Delta \vdash t\ \textsc{duplicate}\ e\ \textsc{times}\ : T[?\textsc{id}]\{\textsc{cols}\}}$$

$$\frac{\Gamma;\Delta \vdash t_1 : T[?\textsc{id}]\{\textsc{cols}\} \quad \Gamma, T[\textsc{id}'']\{\textsc{cols}\};\Delta \vdash t_2 : T[?\textsc{id}']\{\textsc{cols}\} \quad \textsc{id}''\ \text{fresh}}{\Gamma;\Delta \vdash \textsc{with}\ t_1\ \textsc{as}\ \textsc{id}'' : t_2 : T[?\textsc{id}']\{\textsc{cols}\}}$$

$$\frac{\begin{array}{c}\Gamma;\Delta \vdash t : T[?\textsc{id}]\{\textsc{cols}\} \quad \Gamma;\Delta, T[?\textsc{id}]\{\textsc{cols}\} \vdash e_i : \sigma_i \text{ for } 1 \le i \le n \\ \Gamma;\Delta, T[?\textsc{id}]\{\textsc{cols}\} \vdash e'_j : \sigma'_j \quad A_j : \sigma'_j \to \sigma''_j \text{ for } 1 \le j \le m \\ \bar{e}\ \textsc{as}\ \overline{\textsc{col}} := [e_1\ \textsc{as}\ \textsc{col}_1, \dots, e_n\ \textsc{as}\ \textsc{col}_n] \quad \overline{\textsc{col}} \cap \overline{\textsc{col}'} = \emptyset \\ \overline{A(e')}\ \textsc{as}\ \overline{\textsc{col}'} := A_1(e'_1)\ \textsc{as}\ \textsc{col}'_1, \dots, A_m(e'_m)\ \textsc{as}\ \textsc{col}'_m \end{array}}{\begin{array}{c}\Gamma;\Delta \vdash \textsc{group}\ t\ \textsc{by}\ \bar{e}\ \textsc{as}\ \overline{\textsc{col}}\ \textsc{aggregating}\ \overline{A(e')}\ \textsc{as}\ \overline{\textsc{col}'} \\ : T[]\{\textsc{col}_1 : \sigma_1, \dots, \textsc{col}_n : \sigma_n, \textsc{col}'_1 : \sigma''_1, \dots, \textsc{col}'_m : \sigma''_m\}\end{array}}$$

$$\frac{\Gamma;\Delta \vdash m : M[?\textsc{id}]\{\textsc{cols}\} \quad \Gamma;\Delta, M[?\textsc{id}]\{\textsc{cols}\} \vdash c^i : \mathcal{E} \quad \textsc{cols}', \textsc{cols}'' \subseteq \textsc{cols}}{\Gamma;\Delta \vdash \textsc{mutual info}\ (\textsc{id}.\textsc{cols}',\ \textsc{id}.\textsc{cols}'', c^i)\ \textsc{under}\ m : \mathbf{Real}}$$

Fig. 27. Type system for the GenSQL expressions omitted in the main text.

$$\llbracket\, \textsc{dedup}\ t \rrbracket(\gamma,\delta) = \Big(\lambda x.\mathbf{fold}\ (\lambda r, y.\{r\} \cup \mathbf{filter}\ (\lambda r'.r = r')\ y)\ \emptyset\ x\Big)_* \llbracket t \rrbracket(\gamma,\delta)$$

$$\llbracket t_1\ \textsc{union}\ t_2 \rrbracket(\gamma,\delta) = (\lambda x, y.\ x \cup y)_*(\llbracket t_1 \rrbracket(\gamma,\delta) \otimes \llbracket t_2 \rrbracket(\gamma,\delta))$$

$$\llbracket t\ \textsc{duplicate}\ e\ \textsc{times}\ \rrbracket(\gamma,\delta) = \mathbf{let}\ n = \llbracket e \rrbracket(\gamma,\delta)\ \mathbf{in}\ \Big(\lambda y. \bigcup_{1\le i\le n} y\Big)_* \llbracket t \rrbracket(\gamma,\delta)$$

$$\llbracket\, \textsc{with}\ t_1\ \textsc{as}\ \textsc{id}'' : t_2 \rrbracket(\gamma,\delta) = \llbracket t_1 \rrbracket(\gamma,\delta) \gg\!= (\lambda x.\llbracket t_2 \rrbracket(\gamma[\textsc{id}'' \mapsto x],\delta))$$

$$\begin{array}{l}\llbracket\, \textsc{group}\ t : T[?\textsc{id}]\{\textsc{cols}\}\ \textsc{by}\ \bar{e}\ \textsc{as}\ \overline{\textsc{col}}\ \textsc{aggregating}\ \overline{A(e')}\ \textsc{as}\ \overline{\textsc{col}'} \rrbracket(\gamma,\delta) = \\ \Big(\lambda x.\mathbf{let}\ y = \mathbf{map}\ (\lambda r.(\llbracket \bar{e} \rrbracket(\gamma,\delta[\textsc{id} \mapsto r]), \llbracket \overline{e'} \rrbracket(\gamma,\delta[\textsc{id} \mapsto r])))\ x \\ \quad \mathbf{let}\ keys = \mathbf{map}\ (\lambda(a,b).b)\ y \\ \quad \mathbf{let}\ bags = \mathbf{map}\ (\lambda a.(a, \mathbf{fold}\ (\lambda(a',b').\mathbf{if}\ a = a' \mathbf{then}\ \{b'\}\ \mathbf{else}\ \{\})\ \{\}\ y))\ keys \\ \quad \mathbf{map}\ (\lambda(a,s).(a,\bar{A}(s)))\ bags\Big)_* \llbracket t \rrbracket(\gamma,\delta) \end{array}$$

$$\text{where } \llbracket \bar{e} \rrbracket(\gamma,\delta) = (\llbracket e_1 \rrbracket(\gamma,\delta), \dots, \llbracket e_n \rrbracket(\gamma,\delta)) \text{ and } \llbracket \overline{e'} \rrbracket(\gamma,\delta) = (\llbracket e'_1 \rrbracket(\gamma,\delta), \dots, \llbracket e'_m \rrbracket(\gamma,\delta))$$

Fig. 28. Denotational semantics for the GenSQL expressions omitted in the main text.

```
WITH model GIVEN Period_minutes > 2000 AS conditional_model:
SELECT
  AVG(log_p) AS entropy
FROM (
  SELECT - LOG(p) AS log_p FROM (
  SELECT
    PROBABILITY OF Perigee_km AND Apogee_km UNDER
       conditional_model AS p
  FROM
    (GENERATE UNDER conditional_model LIMIT 1000)))
```

(a) Entropy on conditioned model

```
SELECT
  AVG(log_p_over_log_q) AS kl_p_q_estimate
FROM (
  SELECT LOG(p) - LOG(q) AS log_p_over_log_q FROM (
    SELECT
      PROBABILITY OF * UNDER model_p AS p,
      PROBABILITY OF * UNDER model_q AS q,
    FROM
    GENERATE UNDER model_p
    LIMIT 1000))
```

(b) Kullback-Leibler divergence

```
GENERATE
UNDER model_a
  GIVEN (x > 10 * PROBABILITY OF y > 0.9 UNDER model_b)
LIMIT 100

GENERATE
UNDER model_a
LIMIT 100
GENERATIVE JOIN model_b GIVEN *
```

(c) Conditioning a model on output of another

```
SELECT
  PROBABILITY OF Purpose AND Country_of_Operator UNDER model
    GIVEN Apogee_km > 35000 AND Period_miutes = 1440 AS p
    Purpose,
    Country_of_Operator,
FROM (
  SELECT
    Purpose,
    Country_of_Operator,
  FROM data
  GROUP BY Purpose AND Country_of_Operator)
ORDER BY p DESC LIMIT 1
```

(d) Maxiumum-a-posteriori

```
SELECT
  *
FROM data
WHERE
  (PROBABILITY OF bmi UNDER model) >
  (PROBABILITY OF bmi UNDER model GIVEN * EXCEPT bmi)
```

(e) Anomaly detection

```
SELECT
  test_data.apogee,
  test_data.perigee,
  model.period
FROM test_data
GENERATIVE JOIN
  model.period GIVEN Perigee_km, Apogee_km
```

(f) Prediction

```
SELECT
  PROBABILITY OF * UNDER model AS p,
  *
FROM
    GENERATE UNDER model LIMIT 1000
ORDER BY p
LIMIT 5
```

(g) Likely synthetic data

```
GENERATE
UNDER model
GIVEN IPTG = "added" AND Arabinose = "added" AND yfp > 100
LIMIT 10000
```

(h) Conditional synthetic data generation

```
SELECT
  experience,
  PROBABILITY OF experience UNDER model
    GIVEN * EXCEPT experience
      AS probability_experience
FROM data
GENERATIVE JOIN
    model GIVEN *
WHERE probability_experience > 0.95
```

(i) Imputation

```
WITH model GIVEN Period_minutes > 2000 AS conditional_model:
SELECT
  AVG(log_pxy_div_px_py) AS mutual_information
FROM (
  SELECT LOG(pxy) - LOG(px) - LOG(py) AS log_pxy_div_px_py FROM
       (
    SELECT
      PROBABILITY OF Perigee_km AND Apogee_km UNDER
       conditional_model AS pxy,
      PROBABILITY OF Perigee_km UNDER conditional_model AS px,
      PROBABILITY OF Apogee_km UNDER conditional_model AS py
    FROM
      (GENERATE UNDER conditional_model LIMIT 1000)))
```

(j) Conditional mutual information

Fig. 29. Example queries in GenSQL for a variety of data analysis tasks.

logpdf 1

```
SELECT
  PROBABILITY OF Period_minutes = 98.6
  UNDER model GIVEN Country_of_Operator
FROM data
```

logpdf 2

```
SELECT
    PROBABILITY OF Period_minutes = 98.6 AND
                Type_of_Orbit = \"Sun-Synchronous\"
    UNDER model GIVEN Country_of_Operator AND
                    Launch_Mass_kg
FROM data
```

logpdf 3

```
SELECT
    PROBABILITY OF Period_minutes = 98.6 AND
                Type_of_Orbit = \"Sun-Synchronous\" AND
                Contractor = \"Lockheed Martin\"
    UNDER model GIVEN Country_of_Operator AND
                    Launch_Mass_kg AND
                    Inclination_radians
FROM data
```

logpdf 4

```
SELECT
    PROBABILITY OF Period_minutes = 98.6 AND
                Type_of_Orbit = \"Sun-Synchronous\" AND
                Contractor = \"Lockheed Martin\" AND
                Eccentricity = 0.001
    UNDER model GIVEN Country_of_Operator AND
                    Launch_Mass_kg AND
                    Inclination_radians AND
                    Apogee_km
FROM data
```

logpdf 5

```
SELECT
    PROBABILITY OF Period_minutes = 98.6 AND
                Type_of_Orbit = \"Sun-Synchronous\" AND
                Contractor = \"Lockheed Martin\" AND
                Eccentricity = 0.001 AND
                Purpose = \"Communications\"
    UNDER model GIVEN Country_of_Operator AND
                    Launch_Mass_kg AND
                    Inclination_radians AND
                    Apogee_km AND
                    Power_watts
FROM data
```

Fig. 30. Queries from Table 1 in GenSQL.

logpdf 6

```
SELECT
    PROBABILITY OF
        Contractor = \"Microsat Systems Canada Inc\"
    UNDER model GIVEN
        Country_of_Contractor
FROM data
```

logpdf 7

```
SELECT
    PROBABILITY OF
        Inclination_radians = 5.52 AND
        Operator_Owner = \"AMSAT-UK\"
    UNDER model GIVEN
        Launch_Vehicle AND
        Eccentricity
FROM data
```

logpdf 8

```
SELECT
    PROBABILITY OF
        Purpose = \"Earth Observation/Research\" AND
        Period_minutes = 5512.43 AND
        Launch_Vehicle = \"Tsyklon 3\"
    UNDER model GIVEN
        Eccentricity AND
        Dry_Mass_kg AND
        Launch_Mass_kg
FROM data
```

logpdf 9

```
SELECT
    PROBABILITY OF
        longitude_radians_of_geo = 2.19 AND
        Eccentricity = 0.00319 AND
        Inclination_radians = 20.67 AND
        Type_of_Orbit = \"Molniya\"
    UNDER model GIVEN
        Launch_Mass_kg AND
        Launch_Vehicle AND
        Purpose AND
        Launch_Site
FROM data
```

logpdf 10

```
    SELECT
        PROBABILITY OF
            Period_minutes = 19529.87 AND
            Type_of_Orbit = \"Deep Highly Eccentric\" AND
            Launch_Site = \"Kodiak Launch Complex\" AND
            Dry_Mass_kg = 5093.73 AND
            Inclination_radians = 8.17
        UNDER model GIVEN
            Contractor AND
            Launch_Mass_kg AND
            Purpose AND
            Perigee_km AND
            Power_watts
    FROM data
```

Fig. 30. Queries from Table 1 in GenSQL (continued).