Type-Directed Discretization of Probabilistic Programs

KATHERINE WU, Cornell University, USA
JULES JACOBS, ETH Zurich, Switzerland and Jane Street, USA
KEVIN BATZ, University of Münster, Germany
ALEXANDRA SILVA, Cornell University, USA

We study *exact discretization* as a semantics-preserving transformation for recursive, higher-order probabilistic programs with continuous distributions. We target programs where continuous values are compared against finitely many constants, so exact inference reduces to a discrete problem. Our central technical contribution is a non-local, type-directed analysis that infers where continuous values can be partitioned into finitely many observationally relevant regions, then rewrites sampling and comparison behavior over those regions. We call this transformation Slice. Because this construction is global and type-directed, correctness requires reasoning beyond the local syntax: we formalize the transformation and prove soundness for boolean queries using a coupling-style logical relations argument over operational semantics. As an application, transformed programs can be executed by discrete engines such as Dice, Roulette, and Storm. Our empirical evaluation shows two complementary strengths of Slice when paired with discrete backends: it enables *exact inference* for challenging continuous programs that lie beyond the reach of previous exact systems, and, on benchmarks where direct comparison is possible, it is competitive with state-of-the-art exact inference systems for continuous programs.

CCS Concepts: • **Mathematics of computing → Probabilistic representations**; **Probabilistic inference problems**; • **Theory of computation → Operational semantics**.

Additional Key Words and Phrases: probabilistic programming, exact inference, discretization, type systems

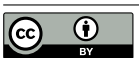
ACM Reference Format:
Katherine Wu, Jules Jacobs, Kevin Batz, and Alexandra Silva. 2026. Type-Directed Discretization of Probabilistic Programs. *Proc. ACM Program. Lang.* 10, OOPSLA2, Article 402 (October 2026), 140 pages. https://doi.org/10.1145/3839534

1 Introduction

Probabilistic programming languages (PPLs) are widely used to model uncertainty [5, 9, 13, 14, 16, 20, 23, 24, 26, 34, 35, 37, 39–41, 45–47]. Programs with *continuous* distributions capture rich models but require reasoning about uncountable domains, whereas purely *discrete* programs often admit fast and precise inference but cannot express real-valued models.

Slice studies *exact discretization*: a semantics-preserving source-to-source transformation that converts, if possible, continuous programs into discrete ones. We target higher-order recursive programs with conditioning and continuous sampling, without arithmetic. The key mechanism is *type-directed cut inference*: our type system tracks each constant threshold (or *cut*) used in comparisons and uses this information to partition continuous domains into finitely many intervals. For example, $\texttt{uniform}(0,1) < 0.5$ discretizes to $\texttt{discrete}(0.5, 0.5) < 1$, yielding a discrete program that existing exact inference tools can process.

Authors' Contact Information: Katherine Wu, Cornell University, Ithaca, USA, kaw324@cornell.edu; Jules Jacobs, ETH Zurich, Zurich, Switzerland and Jane Street, New York, USA, julesjacobs@gmail.com; Kevin Batz, University of Münster, Munster, Germany, kevin.batz@uni-muenster.de; Alexandra Silva, Cornell University, Ithaca, USA, alexandra.silva@cornell.edu.



ACM 2475-1421/2026/10-ART402
https://doi.org/10.1145/3839534

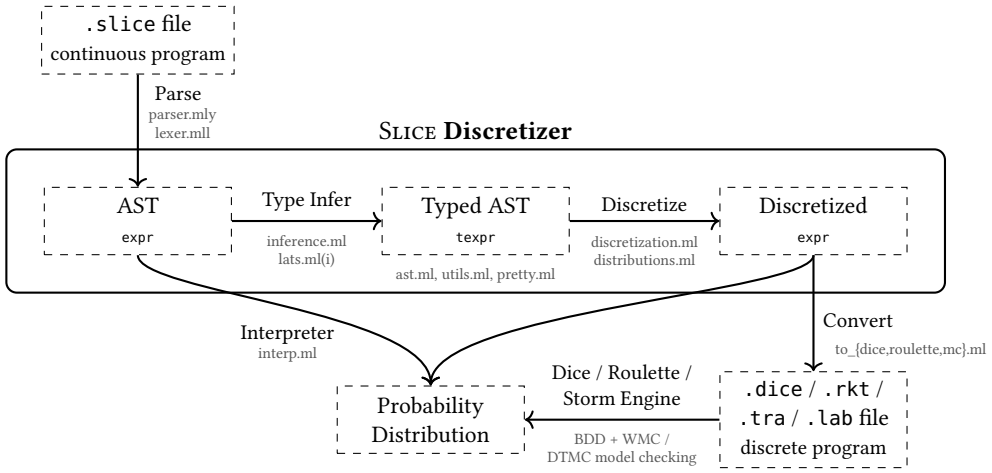


Fig. 1. Pipeline for exact discretization: typed Slice programs are transformed into semantically equivalent discrete programs, then evaluated by discrete engines. The transformation stage (type inference + discretization) is backend-agnostic; Dice, Roulette, and Storm are examples of target backends.

Cuts may arise far from their source expressions: a sampled value may flow through higher-order functions, conditionals, recursive calls, data structures, and observations before it is eventually compared. To discretize soundly, Slice propagates cut information along these non-local flows, collecting all observationally relevant cut points for each continuous value and using them to partition its domain into finitely many intervals (see §2 for illustrative examples). A type system provides a natural framework for this analysis: it gives a compositional account of how cut information propagates through each language construct (via one typing rule per langauge feature); it separates the analysis result, i.e. a type derivation, from the type inference algorithm used to infer it; and it supports our correctness argument, since semantic preservation can be stated as a logical relation indexed by our types carrying cut and value sets.

Our discretization method is subject to a principled restriction: a continuous value must ultimately be compared against constants; direct comparisons between two unknown reals such as $\texttt{uniform}(0,1) < \texttt{uniform}(0,1)$ cannot be discretized exactly and remain continuous. Nonetheless, benchmarks from the literature and our novel case studies show that many programs *can* be discretized by our system automatically.

Discretization itself is not new. Garg et al. [19] encode real-valued variables via user-chosen bit-precisions, trading accuracy for tractability. Other approximate discretization schemes exist in the literature [4, 11, 27]. In Slice, *exact discretization* means that the transformation is semantics-preserving: the discrete program preserves the relevant probabilities of the original continuous program. This differs from approximate discretization, which approximates the original semantics and may provide error bounds or convergence guarantees, but does not prove equality of source and transformed program probabilities in the sense of our main theorem.

Figure 1 shows the concrete pipeline from typed source programs to discrete backends. Because discretization is source-to-source, any discrete PPL that supports finite distributions can serve as a backend; we instantiate this with Dice [26], Roulette [35], and Storm [15].

Outline. Starting from examples (§2) and concluding with related and future work in §8, we:

Type System (§3) Characterize when programs are discretizable.

Type Inference (§4) Infer types automatically to guide discretization.

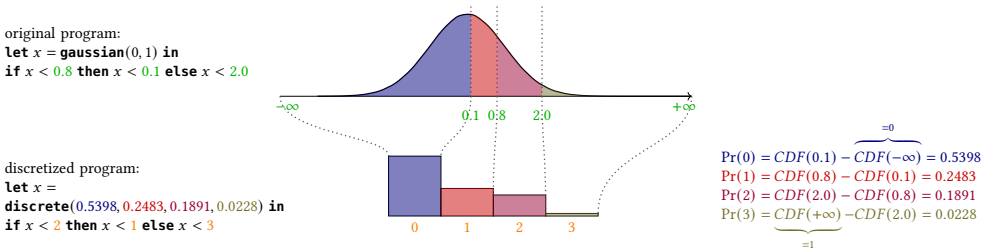


Fig. 2. Discretization of the original program into its semantically equivalent discretized program for §2.1. The shaded regions show how the cuts at 0.1, 0.8, and 2.0 partition the continuous distribution into four intervals $(-\infty, 0.1)$, $[0.1, 0.8)$, $[0.8, 2.0)$, and $[2.0, +\infty)$, whose masses become the probabilities of outcomes 0, 1, 2, and 3, i.e., interval indices, in the discrete distribution.

Exact Discretization (§5) Define a type-directed transformation from continuous programs to discrete ones using inferred type information.

Soundness (§6) Prove semantic preservation of the discretization transformation with a coupling-style relational argument [6, 12, 48].

Implementation, Case Studies, and Evaluation (§7) Implement Slice type inference and the discretization transformation in OCaml, and provide Dice, Roulette, and Storm backends for running the discretized programs. We provide (i) case studies combining continuous sampling, higher-order functions, and unbounded recursion; to the best of our knowledge, there is no prior exact inference system that can cover programs *combining* all three of these features and (ii) case studies where Slice+Dice and Slice+Roulette show favorable scaling and empirical performance on new and existing benchmarks [21, 40].

2 Slice by Example

To build intuition for Slice, we start with simple examples demonstrating the key ideas, and then provide a more involved example containing higher-order recursion and lists in §2.6. Our pipeline performs *fully automatic exact inference* on all provided programs.

2.1 Branching on a Single Continuous Variable

Consider a program that samples a real value from a standard Gaussian distribution and then performs two nested comparisons: first checking whether the sample is less than 0.8, then comparing it to either 0.1 or 2.0:

```
let x = gaussian(0,1) in if x < 0.8 then x < 0.1 else x < 2.0
```

We discretize this program by replacing the real-valued sample with a discrete interval index, as shown below and in Figure 2:

```
let x = discrete(0.5398, 0.2483, 0.1891, 0.0228) in if x < 2 then x < 1 else x < 3
```

Here, **discrete**(0.5398, 0.2483, 0.1891, 0.0228) already represents all four observationally distinguishable outcomes of the Gaussian sample. The precise real value of x never matters to the rest of the program; only the interval in which it falls does. This is the regime we target: continuous quantities observed only through finitely many comparison outcomes. *Slice's goal is to detect such situations automatically.* We partition the continuous distribution into intervals that are indistinguishable to the rest of the program, then map every real to the interval it inhabits.

Our type system guides the transformation, in particular through a *cut set* — the set of comparison thresholds extracted from the program. A continuous distribution is annotated with its cut set, which records the values it is compared against. For our example, the type annotation is:

```
let x = (gaussian(0,1) : float[{<0.1,<0.8,<2}]) in if x < 0.8 then x < 0.1 else x < 2.0
```

Intuitively, the cuts $\{<0.1, <0.8, <2.0\}$ partition the real line $\mathbb{R}$ into four intervals $(-\infty, 0.1)$, $[0.1, 0.8)$, $[0.8, 2.0)$, $[2.0, +\infty)$, and only the interval containing the Gaussian sample x affects the rest of the computation. Slice uses these cuts to replace the continuous distribution with a discrete one whose outcomes correspond exactly to these intervals, and each floating-point constant is mapped to the index of the interval that contains it.

2.2 Branching on Multiple Continuous Variables

Different continuous quantities may induce different cut sets, and Slice computes the appropriate set for each one independently. Consider an example with multiple continuous variables:

```
let x = uniform(0, 1) in
let y = uniform(0, 2) in
if x < 0.5 then x < 0.2 else y < 0.1
```

In this program, the variable x is drawn from **uniform**$(0, 1)$ and compared against two constants: 0.5 and 0.2. These cuts <0.5 and <0.2 partition the real line $\mathbb{R}$ into three intervals: $(-\infty, 0.2)$, $[0.2, 0.5)$, and $[0.5, +\infty)$. Similarly, the variable y, drawn from **uniform**$(0, 2)$, is compared only with 0.1. This single cut <0.1 partitions the real line $\mathbb{R}$ into $(-\infty, 0.1)$ and $[0.1, +\infty)$.

Type inference performs this analysis by collecting all cuts for each continuous variable. For x, it discovers the cuts <0.2 and <0.5, giving it the type `float[{<0.2,<0.5}]`. For y, it only finds <0.1, giving it the type `float[{<0.1}]`. This information is captured in the program below (left):

(a) Type-Annotated Program

```
let x = uniform(0, 1) : float[{<0.2,<0.5}] in
let y = uniform(0, 2) : float[{<0.1}] in
if x < 0.5 then x < 0.2 else y < 0.1
```

(b) Discretized Program

```
let x = discrete(0.2, 0.3, 0.5) in
let y = discrete(0.05, 0.95) in
if x < 2 then x < 1 else y < 1
```

Slice therefore discretizes x into three outcomes **discrete**$(0.2, 0.3, 0.5)$ and y into two outcomes **discrete**$(0.05, 0.95)$, where each probability in the discrete distribution is the measure of the corresponding original interval. Comparisons then operate over the integer indices of those intervals. The constants 0.5 and 0.2 map to indices 2 and 1, respectively, in the partition $(-\infty, 0.2) \cup [0.2, 0.5) \cup [0.5, +\infty)$ for x; the constant 0.1 maps to index 1 in the partition $(-\infty, 0.1) \cup [0.1, +\infty)$ for y.

2.3 Control Flow

We now consider comparisons against a continuous quantity whose value is determined by control flow. Additionally, continuous values can flow through conditionals:

```
let x = if flip() then uniform(0,2) else gaussian(0,1) in
let y = if 1.5 > x then 1.8 else 0.3 in x <= y
```

Here, y evaluates to either 1.8 or 0.3 depending on x. Because $x \leq y$ may use either value, Slice collects both constants and treats the comparison as ranging over this finite set. This motivates enriching our float types with a *value set*: a finite collection of concrete values that an expression

float[cut-set ; value-set]

cut-set: set of cuts (e.g., $\{<0.1, <0.5, \leq 0.8\}$) or $\top$ (uncut)

value-set: set of concrete values (e.g., $\{0.1, 0.4, 0.5\}$) or $\top$ (any value)

Fig. 3. The type **float**[cut-set; value-set] captures two aspects of a floating-point expression. The cut set is determined either by (i) the constants the expression is compared against or (ii) the constant itself if the expression is a float constant. The value set records the concrete values an expression can take. The discretization is driven by the cut set: the n cuts define $n + 1$ intervals over the real line.

may evaluate to. Figure 3 presents the *full* float type implemented by Slice; each floating-point expression is annotated with such a type. The type-annotated program is below (left):

(a) Type-Annotated Program

```
let x =
 (if flip() then
   (uniform(0,2) : float[{<=0.3,<1.5,<=1.8}; T])
 else
   (gaussian(0,1) : float[{<=0.3,<1.5,<=1.8}; T])
 : float[{<=0.3,<1.5,<=1.8}; T]) in
let y =
 (if (1.5 : float[{<=0.3,<1.5,<=1.8}; {1.5}]) > x
   then
     (1.8 : float[{<=0.3,<1.5,<=1.8}; {1.8}])
   else
     (0.3 : float[{<=0.3,<1.5,<=1.8}; {0.3}])
 : float[{<=0.3,<1.5,<=1.8}; {0.3,1.8}]) in
x <= y
```

(b) Discretized Program

```
let x =
 if flip() then
   discrete(0.15, 0.6, 0.15, 0.1)
 else
   discrete(0.617911, 0.315281,
            0.030877, 0.035930)
in
let y =
 if 2 > x then 2 else 0
in
x <= y
```

Value sets refine an expression's cut set by adding cuts induced by comparisons. For example, in the test $x \leq y$, the value set of y contributes boundary points that must also appear in the cut set of x. This dependency is non-local: constants introduced in one branch can constrain how a sampled value from another branch must be discretized.

Whenever two expressions are compared, the type system equates their cut sets so they are discretized over the same intervals. These cut sets then propagate backward through the control flow to all contributing subexpressions. The extended version [51, Figure 18] illustrates this non-local propagation, which the soundness proof must relate through a semantic coupling relation.

The discretized program is shown on the right. The cuts ≤ 0.3, <1.5, ≤ 1.8 induce the intervals $(-\infty, 0.3]$, $(0.3, 1.5)$, $[1.5, 1.8]$, $(1.8, +\infty)$. Each **discrete** statement lists the mass of these intervals rounded to six decimal places.

2.4 Discrete Latent Variable

Continuous distributions can have parameters that depend on discrete random choices. In this example, a Gaussian distribution's standard deviation is determined by such a random choice:

```
let x = if flip() then 0.5 else 1.5 in gaussian(0, x) < 0.5
```

The type annotations show how the variable x determines the Gaussian's standard deviation, and the extended version [51, Figure 19] illustrates how type information is propagated.

(a) Type-Annotated Program

```
let x =
  (if flip() then
      (0.5 : float[{<=0.5,<=1.5}; {0.5}])
    else
      (1.5 : float[{<=0.5,<=1.5}; {1.5}])
  : float[{<=0.5,<=1.5}; {0.5,1.5}])
in
((gaussian(0, x) : float[{<0.5}; T])
  < (0.5 : float[{<0.5}; {0.5}]))
```

(b) Discretized Program

```
let x =
  if flip() then 0 else 1
in
(if x == 0 then
    discrete(0.84, 0.16)
  else
    discrete(0.63, 0.37)) < 1
```

Because x is used as a parameter to the **gaussian** distribution, Slice generates a separate case for each possible value that x can take (namely 0.5 and 1.5). To support this, the type system uses non-strict inequalities in cut sets, allowing these values to map to unique integer indices starting at 0. Specifically, 0.5 maps to index 0 via `float[{<=0.5,<=1.5}; {0.5}]`, and 1.5 maps to index 1 via `float[{<=0.5,<=1.5}; {1.5}]`. The resulting discretized program then branches on these values and instantiates the corresponding discrete distribution, depending on which standard deviation was selected by the initial flip, as shown in the program on the right.

2.5 Conditioning

Slice supports conditioning via **observe**. Consider the following:

(a) Type-Annotated Program

```
let x =
 (uniform(0.0, 1.0) : float[{<0.2,<0.5}; T]) in
let _ =
 observe(x < (0.5 : float[{<0.2,<0.5}; {0.5}])) in
x < (0.2 : float[{<0.2,<0.5}; {0.2}])
```

(b) Discretized Program

```
let x = discrete(0.2, 0.3, 0.5) in
let _ = observe (x < 2) in x < 1
```

In the discretized program, the observation $x < 2$ eliminates the discrete state corresponding to the original interval $[0.5, +\infty)$, thereby preserving the effect of conditioning on $x < 0.5$.

Exact equalities are encoded with **observe** ($x \leq c$ && $x \geq c$). The two cuts $<c$ and $\leq c$ induce the partition $(-\infty, c)$, $\{c\}$, (c, ∞), so Slice can track probability mass assigned exactly to c as its own discrete outcome. This supports point observations when the observed value has strictly positive probability mass; the extended version [51, § A.3] gives such an example.

Measure-zero observations remain delicate [28]. What matters for **observe**(e) is whether e has positive probability, not whether it is written as an equality. When an observation has probability zero, Slice records observation-failure mass instead of renormalizing (cf. §6); if all observation-satisfying executions have probability zero, the result is the empty subdistribution. Scoring semantics for true measure-zero observations are out of scope, so users should approximate them with narrow intervals when that is the intended model. [51, § A.4] gives details and examples.

2.6 Unbounded Recursion, Higher-Order Functions, and Lists

We now turn to the more involved example in Figure 4, which Slice discretizes automatically. This example emphasizes how our discretization procedure operates across higher-order functions, unbounded recursion, and lists.

```
(* Returns true if y is in [lo,hi) *)                    1
let within = fun lo -> fun hi -> fun y ->                 2
  (y >= lo) && (y < hi) in                                3
                                                          4
(* Function for sampling interval bounds *)               5
let b = fun _ ->                                          6
  discrete(0.18: 0.05, 0.11: 0.24,                        7
           0.21: 0.37, 0.09: 0.58,                        8
           0.41: 0.89) in                                 9
                                                          10
(* Initialize intervals by sampling their                 11
respective (possibly overlapping) bounds *)               12
let init =                                                13
  (false, within 0.00 (b ())) ::                          14
  (false, within (b ()) (b ())) ::                        15
  (false, within (b ()) (b ())) ::                        16
  (false, within (b ()) 1.00) ::                          17
  nil in                                                  18
                                                          19
(* Update flags depending on y falling                    20
   within the respective interval *)                      21
let update = fix update y := fun ys ->                    22
  match ys with                                           23
    | nil -> nil                                          24
    | h :: t ->                                           25
        ((fst h || ((snd h) y)), snd h)                   26
        :: (update y t)                                   27
  end in                                                  28
```

```
(* Sample from a gaussian, then use                       29
   the resulting value to update flags *)                 30
let step = fun ys ->                                      31
  update (gaussian(0,1)) ys in                            32
                                                          33
(* Number of samples determined by                        34
   unbounded recursion *)                                 35
let cover = fix cover st :=                               36
  if uniform(0,1) < 0.1 then st                           37
  else cover (step st) in                                 38
                                                          39
(* Have all intervals been covered at                     40
   least once? *)                                         41
let is_covered = fix is_covered ys :=                     42
  match ys with                                           43
    | nil -> true                                         44
    | h :: t -> (fst h) && (is_covered t)                 45
  end in                                                  46
                                                          47
(* Run entire process on sampled intervals *)             48
let final = cover init in                                 49
  is_covered final                                        50
```

Fig. 4. An infinite stochastic process for random interval covering over $[0, 1]$.

The program models the following process: We start by randomly sampling 4 intervals (l. 6 and 13), stored as a list of pairs, where the first component is a flag and the second component is a function that determines whether a given point falls within that interval. We then employ unbounded recursion (l. 36) to repeat the following a potentially unbounded number of times according to a geometric distribution: sample a point y from a gaussian (l. 31), and update the flag of every interval to true whenever y falls within that interval (l. 22). The program returns true iff *all* intervals have been hit at least once (l. 50 and 42). Exact inference on this program therefore determines the probability that this infinite stochastic process hits every interval at least once.

Slice *automatically* produces a *provably equivalent* (cf. §6) discrete program (shown in [51, § A.5]). Even though this program still contains unbounded recursion, our back-end automatically determines that its stochastic execution behavior can be modeled as a *finite-state* Markov chain. Our key insight is that exact inference can then be performed via *probabilistic model checking*: we use Storm [15], which automatically computes that the sought-after probability is 0.034474…. See §7.1 for details. To the best of our knowledge, our pipeline is the first capable of performing exact inference on suitable programs containing continuous sampling, higher-order functions, and unbounded recursion. In §7.1.2, we extend this program by conditioning *inside the recursion*.

3 The Slice Language and Type System

We now describe the language (§3.1), its types (§3.2), and the typing rules for discretization (§3.3). Henceforth, we use the terms *expression*, *subexpression*, and *program* somewhat interchangeably when the intended meaning is clear from context.

$$
\begin{array}{rl}
e ::= x \mid c \mid \text{true} \mid \text{false} \mid k_{\#n} \mid () & \text{constants} \\
\mid \textbf{let}\ x = e_1\ \textbf{in}\ e_2 & \text{let binding} \\
\mid \textbf{uniform}(e_1, e_2) \mid \textbf{gaussian}(e_1, e_2) \mid \textbf{exponential}(e) \mid \ldots & \text{continuous distributions} \\
\mid \textbf{discrete}(p_0, \ldots, p_{n-1}) & \text{discrete distribution} \\
\mid e_1 < e_2 \mid e_1 \le e_2 \mid e_1 <_{\#n} e_2 \mid e_1 \le_{\#n} e_2 & \text{comparisons} \\
\mid e_1 \mathbin{\&\&} e_2 \mid e_1 \mid\mid e_2 \mid \text{not } e & \text{Boolean operations} \\
\mid \textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3 & \text{conditional} \\
\mid (e_1, e_2) \mid \textbf{fst}\ e \mid \textbf{snd}\ e & \text{pairs} \\
\mid \textbf{fun}\ x \to e \mid e_1\ e_2 \mid \textbf{fix}\ f\ x := e & \text{functions} \\
\mid \textbf{observe}\ e \mid e_1; e_2 & \text{effects} \\
\mid \text{nil} \mid e_1 :: e_2 \mid (\textbf{match}\ e\ \textbf{with} \mid \text{nil} \to e_{\text{nil}} \mid e_1 :: e_2 \to e_{\text{cons}}\ \textbf{end}) & \text{lists} \\
\mid \textbf{diverge} & \text{diverge}
\end{array}
$$

Fig. 5. Syntax of the Slice language. Slice supports a variety of continuous distributions with one and two parameters; for a full list of supported distributions, see the extended version [51, § C].

3.1 The Slice Language

The syntax in Figure 5 extends a standard higher-order functional core with probabilistic choices. Beyond constants, variables, comparisons, Boolean operators, conditionals, pairs, and lists, Slice offers first-class functions, recursion, and observations. The construct $\textbf{observe}(e)$ conditions on e holding. Finite-domain constants $k_{\#n}$ range over $\{0_{\#n}, \ldots, (n-1)_{\#n}\}$ as discussed below. The expression $\textbf{discrete}(p_0, \ldots, p_{n-1})$ requires a nonempty list of probabilities: $n \geq 1$, every $p_i \geq 0$, and $\sum_i p_i = 1$. It returns $k_{\#n}$ with probability p_k. For random sampling, we include the usual discrete distributions and any continuous distribution with a tractable cumulative distribution function (one- or two-argument variants such as exponential, laplace, uniform, gaussian, beta, gamma). Adding further distributions is straightforward as long as their CDFs are known; the extended version [51, § C] lists the ones implemented. For brevity, we include only $<$ and $\leq$ (and resp. $<_{\#n}$ and $\leq_{\#n}$) in the language, but we desugar $>$ and $\geq$ (and their finite variants $>_{\#n}$ and $\geq_{\#n}$) by swapping their operands, as well as $==_{\#n}$ by binding each operand once and testing both $\leq_{\#n}$ directions.

3.2 Types

We formalize the types that Slice supports. A *value* set V is a finite set of reals or $\top$. Elements of $\{{<}x \mid x \in \mathbb{R}\} \cup \{{\leq}x \mid x \in \mathbb{R}\}$ are called *cuts*. Cut sets induce intervals partitioning the reals as follows: Given a non-empty cut set $B = \{{\sim_1} x_1, \ldots, {\sim_n} x_n\}$ with $x_1 \leq \ldots \leq x_n$, ordering $<c$ before $\leq c$ when both occur, we define the set $\text{Intervals}(B)$ of intervals it induces as expected, e.g., $\text{Intervals}(\{{<}0.1, {\leq}0.5\}) = \{(-\infty, 0.1), [0.1, 0.5], (0.5, \infty)\}$. The strict and non-strict cuts differ only in which side contains the boundary point: $\text{Intervals}(\{{<}c\}) = \{(-\infty, c), [c, \infty)\}$, whereas $\text{Intervals}(\{{\leq}c\}) = \{(-\infty, c], (c, \infty)\}$. Thus a boundary generated by $e < c$ places c on the right, while a boundary generated by $e \leq c$ places c on the left. Moreover, we let $\text{Intervals}(\emptyset) = \{(-\infty, \infty)\}$ and $\text{Intervals}(\top) = 2^{\mathbb{R}}$. We denote by $\text{Intervals}(B)_i$ the i-th interval of $\text{Intervals}(B)$, assuming $\text{Intervals}(B)$ is 0-indexed, i.e., $i \in \{0, ..., n\}$. We denote the set of all cut- and value sets, respectively, by $\mathcal{B}$ and $\mathcal{V}$.

Definition 3.1 (Types). For $n \in \mathbb{N}$, $B \in \mathcal{B}$, and $V \in \mathcal{V}$, types τ adhere to the grammar

$$\tau ::= \textbf{int} \mid \textbf{bool} \mid \textbf{unit} \mid \textbf{fin}(n) \mid \textbf{float}[B; V] \mid \tau_1 * \tau_2 \mid \tau_1 \to \tau_2 \mid \textbf{list}(\tau)\ .$$

$$\textsc{Float-Const}\ \frac{\mathsf{has}(V, c)}{\Gamma \vdash c : \mathbf{float}[B; V]} \qquad \textsc{Continuous-1}\ \frac{\Gamma \vdash e : \mathbf{float}[B_1; V_1] \quad \mathsf{recovers}(B_1, V_1) \quad B_1 \equiv_\top B}{\Gamma \vdash \mathbf{exponential}(e) : \mathbf{float}[B; \top]}$$

$$\textsc{Continuous-2}\ \frac{\Gamma \vdash e_1 : \mathbf{float}[B_1; V_1] \quad \mathsf{recovers}(B_1, V_1) \quad B_1 \equiv_\top B \quad \Gamma \vdash e_2 : \mathbf{float}[B_2; V_2] \quad \mathsf{recovers}(B_2, V_2) \quad B_2 \equiv_\top B}{\Gamma \vdash \mathbf{uniform}(e_1, e_2) : \mathbf{float}[B; \top]}$$

$$\textsc{Less}\ \frac{\Gamma \vdash e_1 : \mathbf{float}[B; V_1] \quad \Gamma \vdash e_2 : \mathbf{float}[B; V_2] \quad \mathsf{answers}_<(B, V_1, V_2)}{\Gamma \vdash e_1 < e_2 : \mathbf{bool}}$$

$$\textsc{Less-Equal}\ \frac{\Gamma \vdash e_1 : \mathbf{float}[B; V_1] \quad \Gamma \vdash e_2 : \mathbf{float}[B; V_2] \quad \mathsf{answers}_\le(B, V_1, V_2)}{\Gamma \vdash e_1 \le e_2 : \mathbf{bool}}$$

Fig. 6. Typing rules for float types and continuous distributions (**uniform** is a representative for all other two-argument distributions). The notions *recovers* and *answers* are defined in Definitions 3.2 to 3.4. Here Γ is a typing environment of the form $\{x_1 : \tau_1, \ldots, x_n : \tau_n\}$, where $\Gamma, x : \tau_1 = (\Gamma \setminus \{x : \tau \mid \tau \text{ type}\}) \cup \{x : \tau_1\}$.

Standard types cover integers, booleans, ordered pairs, functions, and lists. Finite types $\mathbf{fin}(n)$ contain $\{0_{\#n}, \ldots, (n-1)_{\#n}\}$; the $\#n$-subscript reminds inference backends such as Dice that these values live in a finite domain. The float type $\mathbf{float}[B; V]$ annotates a real-valued expression with its cut set B (an over-approximation of the comparisons it participates in) and value set V (an over-approximation of the values it can take). $B = \top$ means the expression may be compared to arbitrary reals; $B = \emptyset$ means it will never be compared; $V = \top$ indicates an unrestricted result.

3.3 Typing Rules

The typing rules are split into three parts: Rules specific to float types and continuous distributions (Figure 6), standard rules for all other language constructs (Figure 7), and subtyping (Figure 8).

To understand the float typing rules, it is helpful to understand how discretization works. A type **float**[B; V] is discretized as follows: If $B \neq \top$ is a finite set of cuts, then the type is discretized to a type $\mathbf{fin}(n+1)$ where $n+1 = |\mathsf{Intervals}(B)|$ is the number of intervals induced by B. For instance, if $B = \{<0.5, \le 0.8\}$, then the type is discretized to $\mathbf{fin}(3)$, where $0_{\#3}$ represents the interval $(-\infty, 0.5)$, $1_{\#3}$ represents the interval $[0.5, 0.8]$, and $2_{\#3}$ represents the interval $(0.8, \infty)$. If $B = \top$, then no discretization is performed, and the type remains **float**.

To understand the typing rules, keep in mind that the cut sets B need to be sufficiently fine-grained — i.e., contain enough cuts — so that the discretized program has enough information to determine how the original continuous program behaved.

Constants. Rule Float-Const assigns a constant c the type $\mathbf{float}[B; V]$ whenever $\mathsf{has}(V, c)$ holds (i.e., $c \in V$ or $V = \top$). Constants do not constrain the cut set, but recording them in V ensures later comparisons know which values must remain distinguishable.

Continuous Distributions. Rule Continuous-1 types a single-argument distribution when its argument e has type $\mathbf{float}[B; V]$ and $\mathsf{recovers}(B, V)$: We write $B_1 \equiv_\top B_2$ iff $B_1 = \top \iff B_2 = \top$.

Definition 3.2. Let $B \in \mathcal{B}$ and $V \in \mathcal{V}$. We say that $\mathsf{recovers}(B, V)$ if $B = \top$ or ($V \neq \top$ and every interval I induced by B contains at most one value in V, i.e., $\forall I \in \mathsf{Intervals}(B) : |V \cap I| \le 1$).

Intuitively, $\mathsf{recovers}(B, V)$ means that the discretization induced by B is lossless on V: from the resulting interval index, the original value in V can be uniquely recovered. For example, $\mathbf{exponential}(e)$ can be discretized only when e takes finitely many recoverable values. Thus V

must be finite and B must separate those values so the discretized program can determine which one occurred. §5 revisits this invariant. The rule for two-argument distributions is analogous.

Comparisons. For $e < c$ with $e : \mathbf{float}[B_1; \top]$ and $c : \mathbf{float}[B_2; \{c\}]$, the discretized e must still decide the predicate, so B_1 must include $<c$. If c can take any value in V_2, every $<v$ for $v \in V_2$ must appear. Symmetrically, for $c < e$ with $e : \mathbf{float}[B_2; V_2]$, B_2 must contain $\leq c$ so the discretized intervals know on which side of c they fall. For example, $2.3 < e$ requires cuts separating $(-\infty, 2.3]$ from $(2.3, \infty)$; finer partitions are allowed but not required.

In general, the typing rule for $e_1 < e_2$ requires that the expressions share a cut set B, and that their respective value sets V_1, V_2 are related in a way that allows the comparison to be determined after discretization, via the $\text{answers}_<(B, V_1, V_2)$ relation:

Definition 3.3. Let $B \in \mathcal{B}$ and $V_1, V_2 \in \mathcal{V}$. We say that $\text{answers}_<(B, V_1, V_2)$ if (1) $B = \top$, or (2) $B \neq \top$ and $V_2 \neq \top$ and $\forall x \in V_2,\ {<x} \in B$, or (3) $B \neq \top$ and $V_1 \neq \top$ and $\forall x \in V_1,\ {\leq x} \in B$.

To understand this definition, we first want to introduce two different perspectives on the B set. The first perspective is that B specifies a set of intervals, such that values of the expression will be mapped to their respective interval index. The second perspective is that this discretized index *still contains enough information to evaluate all comparisons in B when viewed as predicates.* That is, if B contains $<x$ then the discretized value of $e : \mathbf{float}[B; V]$ still contains enough information to answer $e < x$ precisely: if after discretization, e has an interval index below the interval containing x, then the comparison is true, and otherwise it is false.

Now, if $e_1 : \mathbf{float}[B; V_1]$ and $e_2 : \mathbf{float}[B; V_2]$, then $\text{answers}_<(B, V_1, V_2)$ ensures that the comparison result can still be determined after discretization because:

(1) If $B = \top$, then no discretization is performed, and the comparison can just be performed on the real-valued expression.
(2) If $B \neq \top$ and $V_2 \neq \top$, then the expression is discretized into intervals as determined by B, but in this case the right-hand side can only take on values in V_2. In this case, all $<x$ for $x \in V_2$ must be contained in B, which means that the value of e_2 can be exactly recovered from its interval index. Then, given the previous observation, we can determine the comparison result by comparing the interval index of e_1 to the interval index of e_2.
(3) If $B \neq \top$ and $V_1 \neq \top$, then the situation is similar to the previous case.

If $V_1 = V_2 = \top$, then $\text{answers}_<(B, V_1, V_2)$ (and analogously $\text{answers}_\leq(B, V_1, V_2)$) requires $B = \top$, so neither comparand is discretized. Additionally, we insist that $B_1 = B_2$, which ensures that both comparands will be discretized to the same type, or not discretized at all.

Finally, for non-strict comparisons, the rule is analogous to Definition 3.3:

Definition 3.4. Let $B \in \mathcal{B}$ and $V_1, V_2 \in \mathcal{V}$. We say that $\text{answers}_\leq(B, V_1, V_2)$ if (1) $B = \top$, or (2) $B \neq \top$ and $V_2 \neq \top$ and $\forall x \in V_2,\ {\leq x} \in B$, or (3) $B \neq \top$ and $V_1 \neq \top$ and $\forall x \in V_1,\ {<x} \in B$.

Subtyping. The type system includes a subtyping relation that captures when one type can be replaced by another. The subtyping rules are shown in Figure 8. For float types $\mathbf{float}[B; V]$, subtyping follows the lattice ordering on V: an expression known to range over some set of values can also be considered to range over a superset of those values. For B, we require equality to make sure that the discretized representation of the expression is not changed by subtyping.

All other typing rules are standard and depicted in Figure 7 and in [51, § D], respectively.

4 Type Inference

We now describe our type inference algorithm, whose goal is to annotate each subexpression of a program with a sound type. Specifically, we focus on expressions of type $\mathbf{float}[B; V]$ and show

$$\textsc{Var}\ \frac{}{\Gamma, x : \tau \vdash x : \tau} \qquad \textsc{Let}\ \frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \mathbf{let}\ x = e_1\ \mathbf{in}\ e_2 : \tau_2} \qquad \textsc{If}\ \frac{\Gamma \vdash e_1 : \mathbf{bool} \qquad \Gamma \vdash e_2 : \tau \qquad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \mathbf{if}\ e_1\ \mathbf{then}\ e_2\ \mathbf{else}\ e_3 : \tau}$$

$$\textsc{Observe}\ \frac{\Gamma \vdash e : \mathbf{bool}}{\Gamma \vdash \mathbf{observe}\ e : \mathbf{unit}} \qquad \textsc{Discrete}\ \frac{n \geq 1 \qquad \forall i \in \{0, \ldots, n-1\}.\ p_i \geq 0 \qquad \sum_{i=0}^{n-1} p_i = 1}{\Gamma \vdash \mathbf{discrete}(p_0, \ldots, p_{n-1}) : \mathbf{fin}(n)} \qquad \textsc{Diverge}\ \frac{}{\Gamma \vdash \mathbf{diverge} : \tau}$$

$$\textsc{FinConst}\ \frac{0 \leq k < n}{\Gamma \vdash k_{\#n} : \mathbf{fin}(n)} \qquad \textsc{FinLess}\ \frac{\Gamma \vdash e_1 : \mathbf{fin}(n) \qquad \Gamma \vdash e_2 : \mathbf{fin}(n)}{\Gamma \vdash e_1 <_{\#n} e_2 : \mathbf{bool}} \qquad \textsc{FinLessEq}\ \frac{\Gamma \vdash e_1 : \mathbf{fin}(n) \qquad \Gamma \vdash e_2 : \mathbf{fin}(n)}{\Gamma \vdash e_1 \leq_{\#n} e_2 : \mathbf{bool}}$$

Fig. 7. Typing rules for general language constructs

$$\textsc{Sub-Refl}\ \frac{\tau \in \{\mathbf{int}, \mathbf{bool}, \mathbf{fin}(n), \mathbf{unit}\}}{\tau <: \tau} \qquad \textsc{Sub-Float}\ \frac{B_1 = B_2 \qquad V_1 \sqsubseteq V_2}{\mathbf{float}[B_1; V_1] <: \mathbf{float}[B_2; V_2]} \qquad \textsc{Sub-Pair}\ \frac{\tau_1 <: \tau_1' \qquad \tau_2 <: \tau_2'}{\tau_1 * \tau_2 <: \tau_1' * \tau_2'}$$

$$\textsc{Sub-List}\ \frac{\tau <: \tau'}{\mathbf{list}(\tau) <: \mathbf{list}(\tau')} \qquad \textsc{Sub-Arrow}\ \frac{\tau_1' <: \tau_1 \qquad \tau_2 <: \tau_2'}{\tau_1 \to \tau_2 <: \tau_1' \to \tau_2'} \qquad \textsc{Sub-Type}\ \frac{\tau_1 <: \tau_2 \qquad \Gamma \vdash e : \tau_1}{\Gamma \vdash e : \tau_2}$$

Fig. 8. Subtyping relation. For both cut- and value sets, we let $d \sqsubseteq d'$ iff $\begin{cases} d \subseteq d' & \text{if } d \neq \top \text{ and } d' \neq \top \\ d' = \top & \text{otherwise .} \end{cases}$

how Slice populates the cut set B and value set V, which later determine how and whether the expression can be discretized. Type inference for all other types is standard and follows the classic Hindley-Milner algorithm. For float-typed expressions, we use a two-phase approach: the first phase traverses the program once to generate constraints over cut and value sets; the second phase solves those constraints to produce a fully annotated program.

The distinction between typing and type inference is important. The correctness argument cleanly separates into two pieces: (1) type inference produces a typing derivation matching our typing rules, and (2) given such a typing derivation, discretization is semantics preserving. Thus the proof of discretization does not have to reason directly about the type inference procedure.

4.1 Constraint Generation

We use the constraint language in Figure 9 to express requirements over the type structure and float refinements. We adopt the framework of [38], introducing a new judgment of the form $\Gamma \vdash e : \tau \mid C$, which reads as *in the environment* Γ, *e has type* τ *and produces the set of constraints* C. In this judgment, Γ and e are inputs to the type inference algorithm, while τ and C are outputs: the algorithm infers the type τ for expression e and generates the constraints C that must hold for this typing to be valid. The inference rules formalize which constraints must be generated and later solved for a given expression e to have inferred type τ. Figure 10 gives the constraint generation rules for Slice. We focus on a small but representative subset of constructs in Slice, capturing the key ideas behind constraint generation for the *full* language.

The type inference rules closely mirror the typing rules in Figure 6, with two key differences. First, the inference rules generate a constraint set instead of having those constraints appear in the premises. Second, subtyping is handled in a syntax-directed way: rather than using a general subtyping rule, we bake the subtyping relation into each rule.

$$C ::= \tau_1 <: \tau_2 \mid \mathsf{recovers}(B, V) \mid \mathsf{answers}_<(B, V_1, V_2) \mid \mathsf{answers}_\leq(B, V_1, V_2) \mid \mathsf{has}(V, c)$$
$$\mid B_1 \sqsubseteq B_2 \mid B_1 = B_2 \mid V_1 \sqsubseteq V_2 \mid V_1 = V_2 \mid B_1 \equiv_\top B_2 \mid C_1, C_2$$

Fig. 9. Syntax for our constraint language. Constraints correspond to the semantic requirements in our type system: subtyping ensures type compatibility, $\mathsf{recovers}(B, V)$ ensures that distribution parameters can be recovered after discretization, $\mathsf{answers}_<(B, V_1, V_2)$ and $\mathsf{answers}_\leq(B, V_1, V_2)$ ensure that comparison results can be determined after discretization, and $\mathsf{has}(V, c)$ ensures that constants are properly tracked. $B_1 \equiv_\top B_2$ holds exactly when $B_1 = \top \iff B_2 = \top$.

TFloat-Const
$$\frac{}{\Gamma \vdash c : \mathbf{float}[B; V] \mid \mathsf{has}(V, c)}$$

TIf
$$\frac{\Gamma \vdash e_1 : \mathbf{bool} \mid C_1 \qquad \Gamma \vdash e_2 : \tau_2 \mid C_2 \qquad \Gamma \vdash e_3 : \tau_3 \mid C_3}{\Gamma \vdash \mathbf{if}\ e_1\ \mathbf{then}\ e_2\ \mathbf{else}\ e_3 : \tau_4 \mid C_1, C_2, C_3, \tau_2 <: \tau_4, \tau_3 <: \tau_4}$$

TVar
$$\frac{}{\Gamma, x : \tau \vdash x : \tau \mid \emptyset}$$

TLess
$$\frac{\Gamma \vdash e_1 : \mathbf{float}[B_1; V_1] \mid C_1 \qquad \Gamma \vdash e_2 : \mathbf{float}[B_2; V_2] \mid C_2}{\Gamma \vdash e_1 < e_2 : \mathbf{bool} \mid C_1, C_2, B_1 = B_2, \mathsf{answers}_<(B_1, V_1, V_2)}$$

TContinuous-1
$$\frac{\Gamma \vdash e : \mathbf{float}[B_1; V_1] \mid C}{\Gamma \vdash \mathbf{exponential}(e) : \mathbf{float}[B; V] \mid C, \mathsf{recovers}(B_1, V_1), B_1 \equiv_\top B, V = \top}$$

TContinuous-2
$$\frac{\Gamma \vdash e_1 : \mathbf{float}[B_1; V_1] \mid C_1 \qquad \Gamma \vdash e_2 : \mathbf{float}[B_2; V_2] \mid C_2}{\Gamma \vdash \mathbf{uniform}(e_1, e_2) : \mathbf{float}[B; V] \mid C_1, C_2, \mathsf{recovers}(B_1, V_1), B_1 \equiv_\top B, \mathsf{recovers}(B_2, V_2), B_2 \equiv_\top B, V = \top}$$

TLessEq
$$\frac{\Gamma \vdash e_1 : \mathbf{float}[B_1; V_1] \mid C_1 \qquad \Gamma \vdash e_2 : \mathbf{float}[B_2; V_2] \mid C_2}{\Gamma \vdash e_1 \leq e_2 : \mathbf{bool} \mid C_1, C_2, B_1 = B_2, \mathsf{answers}_\leq(B_1, V_1, V_2)}$$

TLet
$$\frac{\Gamma \vdash e_1 : \tau_1 \mid C_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : \tau_2 \mid C_2}{\Gamma \vdash \mathbf{let}\ x = e_1\ \mathbf{in}\ e_2 : \tau_2 \mid C_1, C_2}$$

Fig. 10. Constraint generation rules for type inference for float-typed expressions.

Example 4.1. To make the constraint generation process concrete, Figure 11 walks through §2.3. Each subexpression is annotated with type variables for the cut sets (B_i) and value sets (V_i), and the constraints produced at each step are shown explicitly.

4.2 Constraint Solving

Once constraint generation completes, we obtain a system of constraints over type variables, cut sets B, and value sets V. The constraint-solving phase proceeds in two steps:

(1) **Subtyping reduction**: We first process all subtyping constraints of the form $\tau_1 <: \tau_2$. Since subtyping for float types follows the rule $\mathbf{float}[B_1; V_1] <: \mathbf{float}[B_2; V_2]$ if and only if $B_1 = B_2$ and $V_1 \sqsubseteq V_2$, we reduce each subtyping constraint to corresponding equality and subset constraints on the cut and value sets. Subtyping elimination continues until only constraints among B's and V's remain.

(2) **Lattice constraint solving**: After subtyping reduction, we are left with constraints purely over cut sets and value sets: equalities, subset relations, and the specialized constraints $\mathsf{recovers}(B, V)$, $\mathsf{answers}_<(B, V_1, V_2)$, $\mathsf{answers}_\leq(B, V_1, V_2)$, and $\mathsf{has}(V, c)$. These constraints are solved using a monotone fixed-point algorithm over the cut and value lattices.

Subtyping Reduction. Subtyping reduction eliminates all remaining type-level subtyping constraints by recursively decomposing them according to type structure. For base float types, a constraint $\mathbf{float}[B_1; V_1] <: \mathbf{float}[B_2; V_2]$ is reduced to the conjunction $B_1 = B_2$ and $V_1 \sqsubseteq V_2$.

Program	Constraints Emitted
`let x = if flip()`	
`then uniform(0 :float`$[B_1; V_1]$`,`	$+ \text{has}(V_1, 0)$
`2 :float`$[B_2; V_2]$`)`	$+ \text{has}(V_2, 2)$
`:float`$[B_3; V_3]$	$+ \text{recovers}(B_1, V_1), B_1 \equiv_\top B_3, \text{recovers}(B_2, V_2), B_2 \equiv_\top B_3, V_3 = \top$
`else gaussian(0 :float`$[B_4; V_4]$`,`	$+ \text{has}(V_4, 0)$
`1 :float`$[B_5; V_5]$`)`	$+ \text{has}(V_5, 1)$
`:float`$[B_6; V_6]$	$+ \text{recovers}(B_4, V_4), B_4 \equiv_\top B_6, \text{recovers}(B_5, V_5), B_5 \equiv_\top B_6, V_6 = \top$
`:float`$[B_7; V_7]$	$+$ `float`$[B_3; V_3] <:$ `float`$[B_7; V_7]$, `float`$[B_6; V_6] <:$ `float`$[B_7; V_7]$
`in`	
`let y = if x :float`$[B_7; V_7]$	$+ \emptyset$
`<`	
`1.5 :float`$[B_8; V_8]$	$+ \text{has}(V_8, 1.5)$
`:bool`	$+ \text{answers}_<(B_7, V_7, V_8), B_7 = B_8$
`then 1.8 :float`$[B_9; V_9]$	$+ \text{has}(V_9, 1.8)$
`else 0.3 :float`$[B_{10}; V_{10}]$	$+ \text{has}(V_{10}, 0.3)$
`:float`$[B_{11}; V_{11}]$	$+$ `float`$[B_9; V_9] <:$ `float`$[B_{11}; V_{11}]$, `float`$[B_{10}; V_{10}] <:$ `float`$[B_{11}; V_{11}]$
`in`	
`x :float`$[B_7; V_7]$	$+ \emptyset$
`<=`	
`y :float`$[B_{11}; V_{11}]$	$+ \emptyset$
`:bool`	$+ \text{answers}_\leq(B_7, V_7, V_{11}), B_7 = B_{11}$

Fig. 11. Program from §2.3 (left) and its emitted constraints from each line (right) from TFloat-Const, TContinuous-2, TIf, TVar, and TLess and TLessEq following the constraint generation rules in Figure 10. Lines annotated with $\emptyset$ emit no constraints.

For compound types, subtyping is handled structurally before reaching floats. For example, a constraint `float`$[B_1; V_1] *$ `float`$[B_2; V_2] <:$ `float`$[B_5; V_5] *$ `float`$[B_6; V_6]$ is decomposed into `float`$[B_1; V_1] <:$ `float`$[B_5; V_5]$ and `float`$[B_2; V_2] <:$ `float`$[B_6; V_6]$, each of which is then reduced to equality and subset constraints on the corresponding cut and value sets. This process is syntactic and continues until no subtyping constraints remain, leaving only constraints over B and V.

Lattice Solving. The lattice solver keeps track of the current contents of all B and V variables. Initially, all B and V variables are set to $\emptyset$. During the solving process, the variables may go up in the lattice to a finite set, or to $\top$. Each of the constraints is handled as follows:

- **Value containment** ($\text{has}(V, c)$): We update V to include the constant c, i.e., $V := V \sqcup \{c\}$. If V is already $\top$, the constraint is trivially satisfied. (1)
- **Top assignment** ($V = \top$): We immediately set V to $\top$, representing that the value set contains all possible float values. (2)
- **Subset constraints** ($B_1 \sqsubseteq B_2$ or $V_1 \sqsubseteq V_2$): We ensure that the contents of the first variable are always a subset of the second. When B_1 or V_1 is updated, we propagate the changes to B_2 or V_2 by taking the least upper bound. (3)
- **Recoverability** ($\text{recovers}(B, V)$): This constraint ensures that the cut set B can distinguish all values in V. If $B = \top$, the constraint is already satisfied. If $V = \top$ and $B \neq \top$, we set $B := \top$, since no finite cut set recovers an unrestriced real value. Otherwise V is finite; we use a heuristic of inserting $\leq x$ for each value $x \in V$. These cuts place distinct elements of V in distinct intervals and therefore satisfy recoverability. This choice can add redundant cuts, but remains sound. (4)
- **Less-than answerability** ($\text{answers}_<(B, V_1, V_2)$): Intuitively, this constraint ensures that B contains enough cuts to answer all possible comparisons of the form "$v_1 < v_2$" where $v_1 \in V_1$ and $v_2 \in V_2$. If $B = \top$, the constraint is trivially satisfied. Otherwise, we again have a heuristic choice:

Subtyping Constraint	After Subtyping Reduction
$\texttt{float}[B_3; V_3] <: \texttt{float}[B_7; V_7]$	$B_3 = B_7,\ V_3 \sqsubseteq V_7$
$\texttt{float}[B_6; V_6] <: \texttt{float}[B_7; V_7]$	$B_6 = B_7,\ V_6 \sqsubseteq V_7$
$\texttt{float}[B_9; V_9] <: \texttt{float}[B_{11}; V_{11}]$	$B_9 = B_{11},\ V_9 \sqsubseteq V_{11}$
$\texttt{float}[B_{10}; V_{10}] <: \texttt{float}[B_{11}; V_{11}]$	$B_{10} = B_{11},\ V_{10} \sqsubseteq V_{11}$

(a) *Subtyping reduction.* Each subtyping constraint over float types is decomposed into an equality constraint on cut sets and a subset constraint on value sets.

	Constraints Processed	Solver Updates
		$\forall i\colon B_i = \emptyset, V_i = \emptyset$
①	$\text{has}(V_1, 0)$	$V_1 := V_1 \sqcup \{0\} = \{0\}$
	$\text{has}(V_2, 2)$	$V_2 := V_2 \sqcup \{2\} = \{2\}$
	$\text{has}(V_4, 0)$	$V_4 := V_4 \sqcup \{0\} = \{0\}$
	$\text{has}(V_5, 1)$	$V_5 := V_5 \sqcup \{1\} = \{1\}$
	$\text{has}(V_8, 1.5)$	$V_8 := V_8 \sqcup \{1.5\} = \{1.5\}$
	$\text{has}(V_9, 1.8)$	$V_9 := V_9 \sqcup \{1.8\} = \{1.8\}$
	$\text{has}(V_{10}, 0.3)$	$V_{10} := V_{10} \sqcup \{0.3\} = \{0.3\}$
②	$V_3 = \top$	$V_3 := \top$
	$V_6 = \top$	$V_6 := \top$
③	$V_3 \sqsubseteq V_7$	$V_7 := V_7 \sqcup \top = \top$
	$V_6 \sqsubseteq V_7$	$\emptyset$
	$V_9 \sqsubseteq V_{11}$	$V_{11} := V_{11} \sqcup \{1.8\} = \{1.8\}$
	$V_{10} \sqsubseteq V_{11}$	$V_{11} := V_{11} \sqcup \{0.3\} = \{0.3, 1.8\}$
④	$\text{recovers}(B_1, V_1)$	$\emptyset$
	$\text{recovers}(B_2, V_2)$	$\emptyset$
	$\text{recovers}(B_4, V_4)$	$\emptyset$
	$\text{recovers}(B_5, V_5)$	$\emptyset$
⑤	$\text{answers}_<(B_7, V_7, V_8)$	$B_7 := B_7 \sqcup \{<1.5\} = \{<1.5\}$
	$\text{answers}_\le(B_7, V_7, V_{11})$	$B_7 := B_7 \sqcup \{\le 0.3, \le 1.8\}$ $= \{<1.5, \le 0.3, \le 1.8\}$
⑥	$B_7 = B_8, B_7 = B_{11}$	$B_7 = B_8, B_7 = B_{11}$
	$B_3 = B_7, B_6 = B_7$	$B_3 = B_7, B_6 = B_7$
	$B_9 = B_{10}, B_{10} = B_{11}$	$B_9 = B_{11}, B_{10} = B_{11}$
⑦	$B_1 \equiv_\top B_3, B_2 \equiv_\top B_3$	$\emptyset$
	$B_4 \equiv_\top B_6, B_5 \equiv_\top B_6$	$\emptyset$

(b) *Lattice solving.* Each pass performs monotone updates to B_i or V_i. Lines annotated with $\emptyset$ denote constraint is trivially satisfied.

Cut Sets

$B_1 = \emptyset$
$B_2 = \emptyset$
$B_4 = \emptyset$
$B_5 = \emptyset$
$B_3 = B_6$
$= B_7 = B_8$
$= B_9 = B_{10}$
$= B_{11}$
$= \{<1.5, \le 0.3, \le 1.8\}$

Value Sets

$V_1 = \{0\}$
$V_2 = \{2\}$
$V_3 = \top$
$V_4 = \{0\}$
$V_5 = \{1\}$
$V_6 = \top$
$V_7 = \top$
$V_8 = \{1.5\}$
$V_9 = \{1.8\}$
$V_{10} = \{0.3\}$
$V_{11} = \{0.3, 1.8\}$

(c) Final cut sets and value sets obtained.

Fig. 12. Following Figure 11, the constraint solving process for §2.3.

we add a cut $<v$ for each value $v \in V_2$ (if $V_2 \neq \top$) and $\le v$ for each value $v \in V_1$ (if $V_1 \neq \top$). If V_1 and V_2 are both $\top$, we set B to $\top$. (5)

- **Less-equal answerability** ($\text{answers}_\le(B, V_1, V_2)$): Analogous to less-than. (5)
- **Equality constraints** ($B_1 = B_2$ or $V_1 = V_2$): We unify the variables, treating them as the same variable throughout the solving process. Any update to one variable is immediately reflected in all variables unified with it. (6)
- **Top consistency** ($B_1 \equiv_\top B_2$): If either variable becomes $\top$, set both to $\top$. (7)

The solver repeatedly processes constraints until reaching a fixed point, where no constraint causes any further updates to the variables. The monotonicity of the lattice operations ensures that

this process terminates. This procedure may add more cuts to Bs than strictly necessary, but it never sets $B = \top$ unless that is the only way to satisfy all constraints. In other words, the discretization itself may be suboptimal in terms of the number of cuts.

Lattice constraint solving takes time polynomial in the size of the program: at worst, *every* constant in the program gets added to the cut set or value set of every subexpression of type $\mathbf{float}[B; V]$.

Additional Constraint for Contextual Equivalence. When the return type of the program involves a float type, either directly or as part of a larger compound type (e.g., $\mathbf{float}[B; V] \to \mathbf{float}[B'; V']$), we add an additional constraint to ensure that these float types are not discretized. We recursively traverse the return type and set all nested B sets to $\top$. This ensures that the observable behavior of the program is preserved after discretization, regardless of the context in which the program is used. When the return type is basic, such as **bool**, nothing needs to be done.

Example 4.2. To make the constraint solving process concrete, Figure 12 walks through §2.3. Each constraint obtained from the generation phase is processed by the solver, then collected to obtain the final cut and value sets.

5 Discretization

This section presents local transformation rules for our type-directed discretization algorithm, with examples. Our goal is to compute, from a typed expression $e : \mathbf{bool}$ and its type annotations, a *discretized* expression $\mathcal{D}[\![e : \mathbf{bool}]\!] : \mathbf{bool}$ such that e and $\mathcal{D}[\![e : \mathbf{bool}]\!]$ are *semantically equivalent*, i.e., they induce the same distribution. We refer to this semantics-preserving transformation as *exact discretization*, to distinguish it from approaches [19, 27] that approximate continuous programs via externally chosen precision parameters. Exactness here is the equality proved by Theorem 6.2: the transformed program preserves the relevant probabilities of the source program, not merely an approximation with an error bound. Intuitively, when a program uses continuous quantities only through finitely many comparison outcomes, exact inference should reduce to a discrete problem, and exact discretization makes that reduction explicit.

As shown in Figure 13, $\mathcal{D}[\![e : \tau]\!]$ is defined recursively on $e : \tau$ and its type annotations; we focus on continuous uniform distributions, with others handled analogously. The only sub-expressions replaced by $\mathcal{D}[\![\cdot]\!]$ are float constants and sampling instructions. The rule schemata are local, but their selection and parameters come from globally inferred cut/value information; §6 proves that this non-local, type-mediated construction preserves semantics. Before detailing the rules, we outline how type annotations guide discretization. The type system is designed so that every typed sub-expression of the form $e = \mathbf{uniform}(e_1, e_2)$ of type $\mathbf{float}[B; V]$ with $B = \{\sim_1 b_1, \ldots, \sim_n b_n\} \neq \top$ can be *discretized* to an expression $e' : \mathbf{fin}(n+1)$ such that the probability that e evaluates to a value in $\mathrm{Intervals}(B)_i$ *coincides* with the probability that e' evaluates to $i_{\#n+1}$. In particular, if *all* $\mathbf{float}[B; V]$-typed subexpressions are finite-cut, then the given expression can be *fully discretized*.

Example 5.1. Consider a uniform distribution whose outcomes are compared to constants:

```
let x = uniform(0, 1) : float[B; T] in
  (x : float[B; T] < (0.3 : float[B; {0.3}])) ||
  (x : float[B; T] >= (0.5 : float[B; {0.5}]))
```

where $B = \{<0.3, <0.5\}$. The cut set B informs us that the probability of the comparisons evaluating to true is determined by the probability of x evaluating to one of the intervals

$$\mathrm{Intervals}(B)_0 = (-\infty, 0.3) \qquad \mathrm{Intervals}(B)_1 = [0.3, 0.5) \qquad \mathrm{Intervals}(B)_2 = [0.5, \infty)$$

induced by B. Hence, all expressions of type `float[B; ·]` can be discretized to expressions of the finite type $\mathbf{fin}(3)$, where $k_{\#3}$ represents the k-th interval $\mathrm{Intervals}(B)_k$ induced by B for $k \in \{0, 1, 2\}$.

$e : \tau$	Condition	$\mathcal{D}[\![e : \tau]\!]$
$x : \tau$		x
$c : \texttt{float}[B;V]$	$B = \top$	c
	$B \neq \top$	$k_{\#(\lvert B\rvert+1)}$ such that $c \in \text{Intervals}(B)_k$
$\texttt{let}\ x = e_1 : \tau_1\ \texttt{in}\ e_2 : \tau_2 : \tau_2$		$\texttt{let}\ x = \mathcal{D}[\![e_1 : \tau_1]\!]\ \texttt{in}\ \mathcal{D}[\![e_2 : \tau_2]\!]$
$(\texttt{if}\ e_1 : \texttt{bool}\ \texttt{then}\ e_2 : \tau\ \texttt{else}\ e_3 : \tau) : \tau$		$\texttt{if}\ \mathcal{D}[\![e_1 : \texttt{bool}]\!]\ \texttt{then}\ \mathcal{D}[\![e_2 : \tau]\!]\ \texttt{else}\ \mathcal{D}[\![e_3 : \tau]\!]$
$(e_1 : \texttt{float}[B;V_1] < e_2 : \texttt{float}[B;V_2]) : \texttt{bool}$	$B \neq \top$	$\mathcal{D}[\![e_1 : \texttt{float}[B;V_1]]\!] <_{\#(\lvert B\rvert+1)} \mathcal{D}[\![e_2 : \texttt{float}[B;V_2]]\!]$
	$B = \top$	$\mathcal{D}[\![e_1 : \texttt{float}[B;V_1]]\!] < \mathcal{D}[\![e_2 : \texttt{float}[B;V_2]]\!]$
$(e_1 : \texttt{float}[B;V_1] \leq e_2 : \texttt{float}[B;V_2]) : \texttt{bool}$	$B \neq \top$	$\mathcal{D}[\![e_1 : \texttt{float}[B;V_1]]\!] \leq_{\#(\lvert B\rvert+1)} \mathcal{D}[\![e_2 : \texttt{float}[B;V_2]]\!]$
	$B = \top$	$\mathcal{D}[\![e_1 : \texttt{float}[B;V_1]]\!] \leq \mathcal{D}[\![e_2 : \texttt{float}[B;V_2]]\!]$
$\texttt{uniform}(e_1 : \texttt{float}[B_1;V_1],\ e_2 : \texttt{float}[B_2;V_2]) : \texttt{float}[B;V]$	$e_1, e_2 \in \mathbb{R},\ e_1 \geq e_2$	$\texttt{diverge}$
	$B = \top$	$\texttt{uniform}(\mathcal{D}[\![e_1 : \texttt{float}[B_1;V_1]]\!], \mathcal{D}[\![e_2 : \texttt{float}[B_2;V_2]]\!])$
	$e_1, e_2 \in \mathbb{R}$, $B = \{\sim_1 b_1, \ldots, \sim_n b_n\}$	$\texttt{discrete}(p_0, \ldots, p_n)$ where $p_k = \text{CDF}(b_{k+1}) - \text{CDF}(b_k)$
	$B \neq \top$, $B_1 \neq \top$, $B_2 \neq \top$, $V_1 = \{v_1, \ldots, v_n\}$, $V_2 = \{w_1, \ldots, w_m\}$	$\texttt{let}\ v_1' = \mathcal{D}[\![e_1 : \texttt{float}[B_1;V_1]]\!]\ \texttt{in}$ $\texttt{let}\ v_2' = \mathcal{D}[\![e_2 : \texttt{float}[B_2;V_2]]\!]\ \texttt{in}$ $\texttt{if}\ v_1' ==_{\#(\lvert B_1\rvert+1)} \mathcal{D}[\![v_1 : \texttt{float}[B_1;V_1]]\!]$ $\quad \&\&\ v_2' ==_{\#(\lvert B_2\rvert+1)} \mathcal{D}[\![w_1 : \texttt{float}[B_2;V_2]]\!]\ \texttt{then}$ $\quad\quad \mathcal{D}[\![\texttt{uniform}(v_1, w_1) : \texttt{float}[B;V]]\!]$ $\ldots$ (enumerate over $V_1 \times V_2$) $\ldots$ $\texttt{else if}\ v_1' ==_{\#(\lvert B_1\rvert+1)} \mathcal{D}[\![v_n : \texttt{float}[B_1;V_1]]\!]$ $\quad \&\&\ v_2' ==_{\#(\lvert B_2\rvert+1)} \mathcal{D}[\![w_m : \texttt{float}[B_2;V_2]]\!]\ \texttt{then}$ $\quad\quad \mathcal{D}[\![\texttt{uniform}(v_n, w_m) : \texttt{float}[B;V]]\!]$ $\texttt{else}\ \texttt{diverge}$

Fig. 13. Discretization function, where $\text{CDF}(b_0) = 0$ and $\text{CDF}(b_{n+1}) = 1$. For brevity, the table writes $\text{CDF}(b_i)$ for each cut b_i, by which we mean either $\text{CDF}(b_i^-)$ or $\text{CDF}(b_i)$ depending on if the cut is strict or non-strict. Here, $\text{CDF}(b_i^-) = \Pr[X < b_i]$ denotes the left limit at b_i, while $\text{CDF}(b_i) = \Pr[X \leq b_i]$. These coincide for continuous distributions, but may differ for discrete or mixed distributions when $\Pr[X = b_i] > 0$. For the **uniform** distribution, $V = \top$ is implicit, and all remaining cut-set and value-set cases are ruled out by type inference. For the final **uniform** case, if $V_1 = \emptyset$ and/or $V_2 = \emptyset$, the case enumeration is empty and thus yields **diverge**. For all other language constructs, $\mathcal{D}[\![\cdot]\!]$ recurses structurally over them.

Now consider the discretized expression obtained from computing $\mathcal{D}[\![e : \textbf{bool}]\!]$, given by

```
let x = discrete(0.3, 0.2, 0.5) in (x <#3 1#3) || (x >=#3 2#3).
```

The probabilities appearing in **discrete**$(\ldots)$ are determined by the cumulative distribution function (CDF) of the uniform distribution over $[0, 1]$, ensuring that the probability of x evaluating to $k_{\#3}$ in $\mathcal{D}[\![e : \textbf{bool}]\!]$ *coincides* with the probability of x evaluating to a value in $\text{Intervals}(B)_k$ in e. Hence, the constants 0.3 and 0.5 in e can be replaced by the constants $1_{\#3}$ and $2_{\#3}$ in $\mathcal{D}[\![e : \textbf{bool}]\!]$.

Crucially, the *answers* constraint in the premise of the typing rules for comparisons (cf. Figure 6) ensures that B is a sufficiently fine-grained partition of $\mathbb{R}$, i.e., there is an injective function $\text{ToInts} : \{0.3, 0.5\} \to \text{Intervals}(B)$ that maps each constant $c \in \{0.3, 0.5\}$ to the unique interval in $\text{Intervals}(B)$ containing c. This interval is determined by the cut convention: a cut $<c$ puts c in the interval to the right of the boundary, while a cut $\leq c$ puts c in the interval to the left.

Example 5.2. If the arguments to a uniform sampling are non-constant, the situation becomes more challenging. Consider the following typed expression e : **bool** (again only showing relevant types):

```
let x = (
  if uniform(0, 1) : float[{<0.5}; T] < (0.5 : float[{<0.5}; {0.5}]) then
    10 : float[{<=10,<=20}; {10}]
  else
    20 : float[{<=10,<=20}; {20}]) : float[{<=10,<=20}; {10,20}] in
uniform(0 : float[{<=0}; {0}], x : float[{<=10,<=20}; {10,20}]) : float[{<5}; T]
  < (5 : float[{<5}; {5}])
```

Why can this expression be discretized even though the argument to the second uniform sampling instruction is non-constant? The typing information (x : `float[{<=10,<=20}; {10,20}]`) correctly tells us that x *can only take the values* 10 *and* 20. Consequently, x (and the **if**-expression generating it) can be discretized to a finite type **fin**(3) such that the probability of the resulting discrete expression evaluating to $0_{\#3}$ (resp. $1_{\#3}$) *coincides* with the probability of the continuous expression evaluating to 10 (resp. 20). Let us (partially) apply $\mathcal{D}[\![e : \textbf{bool}]\!]$, resulting in:

```
let x = if discrete(0.5, 0.5) <#2 1#2 then 0#3 else 1#3 in
( let v1' = D[[0 : float[{<=0}; {0}]]] in
  let v2'  = D[[x : float[{<=10,<=20}; {10,20}]]] in
  if v1' ==#2 D[[0 : float[{<=0}; {0}]]] &&
     v2' ==#3 D[[10 : float[{<=10,<=20}; {10,20}]]]
  then D[[uniform(0,10) : float[{<5}; T]]]
  else D[[uniform(0,20) : float[{<5}; T]]] ) <#2 D[[5 : float[{<5}; {5}]]]
```

The continuous $\textbf{uniform}(0, x)$-expression is replaced by a case distinction on the discretizations of $\textbf{uniform}(0, 10)$ and $\textbf{uniform}(0, 20)$, with each case executed with the correct probability 0.5. Since this reduces the remaining discretization process to discretizing expressions with *constant parameters only*, the remaining steps are completely analogous to Theorem 5.1.

Crucially, the *recoverability* constraints in the Continuous-rules from Figure 6 ensure that the parameter expressions of the respective sampling expressions can take only finitely many values, and that the cut sets are fine-grained enough to recover the values the parameter expressions evaluate to, as demonstrated in the preceding example.

A similar intuition applies to discretizable comparisons $e_1 < e_2$, where *both* expressions are non-constant but finite-cut. The *answers* constraints in the rule Less (resp. Less-Equal) ensure that at least one of e_1 and e_2 evaluates to only finitely many values, which enables discretization:

Example 5.3. Consider the following typed expression $(e_1 \leq e_2)$: **bool**, given by

```
(if discrete(0.5, 0.5) <#2 1#2 then
  uniform(0,10) : float[B; T] else 2.5 : float[B; {2.5}])
<=
(if discrete(0.5, 0.5) <#2 1#2
  then 2.5 : float[B; {2.5}] else 3.5 : float[B; {3.5}])
```

where $B = \{\leq 2.5, \leq 3.5\}$, and where e_1 : `float[B; T]` and e_2 : `float[B; {2.5,3.5}]`. Intuitively, e_1 samples from a mixture of a uniform and a dirac distribution, and e_2 samples from a finite distribution. Why can $e_1 \leq e_2$ be discretized even though neither e_1 nor e_2 is a constant? As suggested by its type, e_2 can take only finitely many values, namely $\{2.5, 3.5\}$. We apply $\mathcal{D}[\![\cdot]\!]$:

```
(if discrete(0.5, 0.5) <#2 1#2 then discrete(0.25, 0.1, 0.65) else 0#3)
<=
(if discrete(0.5, 0.5) <#2 1#2 then 0#3 else 1#3)
```

$$\mathcal{D}[\![\tau]\!] = \begin{cases} \mathbf{fin}(|B|+1) & \text{if } \tau = \mathbf{float}[B;V] \text{ and } B \neq \top \\ \mathbf{float}[B;V] & \text{if } \tau = \mathbf{float}[B;V] \text{ and } B = \top \\ \text{(straightforward recursion on type structure)} & \text{otherwise} \end{cases}$$

Fig. 14. Transformation of types under $\mathcal{D}$, where $\mathcal{D}$ acts on types.

Notice that the probability of e_1 evaluating to some value in $(-\infty, 2.5]$ indeed *coincides* with the probability of the discretization of e_1 evaluating to $0_{\#3}$. For e_2, on the other hand, the probability of it evaluating to 2.5 (resp. 3.5) *coincides* with the probability of its discretization evaluating to $0_{\#3}$ (resp. $1_{\#3}$). For this reason, we obtain a semantically equivalent discretization of $e_1 \leq e_2$.

Discretization of Types. We conclude by extending $\mathcal{D}[\![\cdot]\!]$ to types. Figure 14 shows how type τ is discretized to type $\mathcal{D}[\![\tau]\!]$. We then obtain the following natural result, proved in [51, § E.1]:

Theorem 5.4 (Discretization Is Well-Typed). *Let $\Gamma \vdash e : \tau$. We have $\mathcal{D}[\![\Gamma]\!] \vdash \mathcal{D}[\![e : \tau]\!] : \mathcal{D}[\![\tau]\!]$, where $\mathcal{D}[\![\Gamma]\!]$ transforms all types in Γ, i.e., $\mathcal{D}[\![\Gamma]\!](x) = \mathcal{D}[\![\Gamma(x)]\!]$ for all $x \in \mathrm{dom}(\Gamma)$.*

6 Operational Semantics and Soundness

We now establish the soundness of our approach in the following sense: For *every* expression $e : \mathbf{bool}$, possibly including *continuous sampling, higher-order functions, recursion, and conditioning*,

$$e : \mathbf{bool} \textit{ and } \mathcal{D}[\![e : \mathbf{bool}]\!] : \mathbf{bool} \textit{ (obtained from Figure 13) are semantically equivalent.}$$

The first step is to make formal sense of semantic equivalence. For that, we will define a monadic operational semantics for our language in §6.1. We will then proceed by proving the above claim. This is non-trivial: discretization decisions are driven by non-local, type-propagated information, so we cannot proceed by a local rule-by-rule argument. To prove the claim rigorously, we relate the executions of e and $\mathcal{D}[\![e]\!]$ through a coupling-style logical relation in §6.1.

6.1 Operational Semantics

The type-directedness of our discretization function $\mathcal{D}[\![\cdot]\!]$ requires our soundness argument to depend on the types of the given (sub-)expressions as well. We thus start by defining a *type-dependent semantic functional*, i.e., we define for every type τ a function $\langle\cdot\rangle : Expr_\tau \to \mathcal{D}(Vals_\tau + \bot)$, mapping each *fully type-annotated* expression $e : \tau$ to a *sub*-probability measure $\langle e : \tau\rangle$ over values of type τ or $\bot$. In the presence of possibly unbounded recursion *and* conditioning, our semantics distinguishes between non-termination and observation failure, which we realize as follows: The "missing" probability mass in $\langle e : \tau\rangle$ is the probability that e diverges. The probability of $\bot$ is the probability of encountering an observation failure. With this model, semantic equivalence indeed captures our sought-after correctness claim: If $\langle e : \mathbf{bool}\rangle = \langle \mathcal{D}[\![e : \mathbf{bool}]\!] : \mathbf{bool}\rangle$, then both expressions have the same probability of producing true, false, or an observation failure.

Remark 6.1. Distinguishing between non-termination and observation failure yields a more fine-grained semantics which does, e.g., *not* semantically equate the two expressions

$$\mathbf{let}\ x = \mathbf{uniform}(0,1)\ \mathbf{in}\ \mathbf{observe}(x > 1) \qquad \text{and} \qquad \mathbf{diverge}\,.$$

In particular, our soundness theorem does not deem the latter a valid discretization of the former.

6.1.1 Small Step Semantics. The type-dependent semantic functional $\langle\cdot\rangle$ is defined via a *small-step* semantic functional $[\![\cdot]\!] : Expr_\tau \to \mathcal{D}(Expr_\tau + \bot)$. Defining the latter requires us to carefully design a suitable measurable space over the set of type-annotated expressions of type τ and $\bot$. All details

on these measure-theoretic constructions are presented in the extended version [51, § F.2]. Let us consider the key idea here.

Let $\tau = \mathbf{bool}$ for the sake of concreteness. Care must be taken since, e.g., sub-expressions of type $\mathbf{float}[B; \{3.5, 4.5\}]$ cannot evaluate to *arbitrary* reals but are restricted to the values $\{3.5, 4.5\}$. We tackle this by extracting from each type-annotated expression a suitable *skeleton*, where float constants are replaced by "holes" (hence the term "skeleton"). For instance, the expression

$$e \quad = \quad (x : \mathbf{float}[\{<3.5, <4.5\}; \top] \ < \ y : \mathbf{float}[\{<3.5, <4.5\}; \{3.5, 4.5\}]) : \mathbf{bool}$$

gives rise to the skeleton

$$s \quad = \quad (\square : \mathbf{float}[\{<3.5, <4.5\}; \top] \ < \ \square : \mathbf{float}[\{<3.5, <4.5\}; \{3.5, 4.5\}]) : \mathbf{bool}$$

and the type annotations tell us that the possible values these holes can be filled with are in $\mathbb{R} \times \{3.5, 4.5\}$. We can thus inherit a measurable space over all well-typed expressions arising from this particular skeleton s from the standard Borel measurable space over $\mathbb{R} \times \{3.5, 4.5\}$. The measurable space over *all* well-typed expressions of type $\mathbf{bool}$ (i.e., over $Expr_{\mathbf{bool}}$) is then obtained from the measurable spaces of *all* skeletons that give rise to expressions of type $\mathbf{bool}$.

Equipped with this family of measurable spaces, we define the small-step functional $[\![\cdot]\!]$ in the expected, type-preserving manner (see [51, Tables 1 and 2]). For instance, if e_1 is not a value, then[1]

$$[\![(\mathbf{let}\ x = e_1 : \tau_1\ \mathbf{in}\ e_2 : \tau_2) : \tau_2]\!] \ = \ \underbrace{[\![e_1 : \tau_1]\!] \gg\!= \lambda g. \begin{cases} \delta_\bot & \text{if } g = \bot \\ \delta_{(\mathbf{let}\ x = g\ \mathbf{in}\ e_2 : \tau_2) : \tau_2} & \text{otherwise} \end{cases}}_{\text{standard probabilistic monadic bind over suitable sub-Markov kernels}},$$

i.e., $[\![e_1 : \tau_1]\!]$ yields the sub-distribution over expressions obtained from executing e_1 for *one step*, and the monadic bind $\gg\!=$ "folds" this distribution into the surrounding $\mathbf{let}$-statement. The intuition for the remaining statements is analogous. Crucially, $\gg\!=$ propagates the probability of encountering $\bot$ — an observation failure — through executions. For instance, assume that e_1 from above is given by $\mathbf{observe}\ (\text{false} : \mathbf{bool}) : \mathbf{unit}$. Since e_1 encounters an observation failure with probability 1, i.e., $[\![\mathbf{observe}\ (\text{false} : \mathbf{bool}) : \mathbf{unit}]\!](\{\bot\}) = 1$, the same holds for the surrounding $\mathbf{let}$-statement, i.e., also $[\![(\mathbf{let}\ x = e_1 : \tau_1\ \mathbf{in}\ e_2 : \tau_2) : \tau_2]\!](\{\bot\}) = 1$. All details — including proofs that all monadic binds are well-behaved since they operate on suitable sub-Markov kernels — are provided in the extended version [51, § F.4].

6.1.2 From Small-Step to Big-Step Semantics. We now define $\langle\cdot\rangle$ via $[\![\cdot]\!]$. For that, we first lift the one-step semantics $[\![\cdot]\!]$ to an n-step semantics $\langle\cdot\rangle_n$ in the expected way:

$$\langle e : \tau\rangle_0 = \delta_{e:\tau} \qquad \text{and} \qquad \langle e : \tau\rangle_{n+1} \ = \ \langle e : \tau\rangle_n \gg\!= \lambda e'. \begin{cases} \delta_\bot & \text{if } e' = \bot \\ [\![e' : \tau]\!] & \text{otherwise} \end{cases}$$

This construction is well-defined since the functional $[\![\cdot]\!]$ itself gives rise to a suitable sub-Markov kernel, as proved in the extended version [51, Lemma F.39]. We then define for every type τ the functional $\langle\cdot\rangle\colon Expr_\tau \to \mathcal{D}(Vals_\tau + \bot)$ as $\langle e : \tau\rangle = \lambda A.\, \lim_{n\to\infty} \langle e : \tau\rangle_n(A)$, which is well-defined by the monotone convergence theorem: the sequence $(\langle e : \tau\rangle_n)_{n\in\mathbb{N}}$ is monotonically increasing for every measurable set A over $Vals_\tau + \bot$ (see the extended version [51, Lemma F.49]) and upper-bounded by 1: executing a program for more and more steps can only increase the probability of terminating in a given set of values or an observation failure.

[1] Put formally, $(\mu \gg\!= f)(A) = \int f(x)(A) \cdot \mu(dx)$; see the extended version [51, Definition F.28] for details. δ_e denotes the Dirac distribution of e.

6.2 Soundness via Probabilistic Couplings

We first state the central theorem, then outline our coupling-style correctness argument. Throughout, $e : \tau$ denotes a closed, fully annotated expression satisfying $\emptyset \vdash e : \tau$ under the declarative typing rules.

Theorem 6.2. *Let* $e : \mathbf{bool}$. *Then* $\langle e : \mathbf{bool} \rangle = \langle \mathcal{D}[\![e : \mathbf{bool}]\!] : \mathbf{bool} \rangle$.

Intuitively, we prove that the two programs can execute in a synchronized manner such that their probabilities of terminating in value v coincide for each $v \in \{\text{true}, \text{false}, \bot\}$.

Towards this end, define for every type τ the set $P_\tau = \{(e : \tau, \mathcal{D}[\![e : \tau]\!] : \mathcal{D}[\![\tau]\!]) \mid e : \tau\}$. We define a *coupled small-step semantics* $[\![\cdot, \cdot]\!] \colon P_\tau \to \mathcal{D}(P_\tau + \bot)$ (see [51, Tables 3 and 4]), which models the simultaneous execution of e and $\mathcal{D}[\![e : \tau]\!]$. The key property of this coupled semantics is that the semantics of both e and $\mathcal{D}[\![e : \tau]\!]$ can be recovered in a sense we formalize next.

Given $\mu \in \mathcal{D}(P_\tau + \bot)$, we define the *projection of μ onto the i-th component* for $i \in \{1, 2\}$ as

$$\pi_i(\mu) = \lambda A. \int_{x \in P_\tau + \bot} \begin{cases} \delta_{e_i}(A) & x = (e_1, e_2) \\ \delta_\bot(A) & x = \bot \end{cases} \mu(dx) \,,$$

With projections, we recover the semantics of e and $\mathcal{D}[\![e : \tau]\!]$ as follows:

Lemma 6.3. *Let* $e : \tau$. *We have:*

(1) $\pi_1([\![e : \tau, \mathcal{D}[\![e : \tau]\!] : \mathcal{D}[\![\tau]\!]]\!]) = [\![e : \tau]\!]$.
(2) $\pi_2([\![e : \tau, \mathcal{D}[\![e : \tau]\!] : \mathcal{D}[\![\tau]\!]]\!]) = [\![\mathcal{D}[\![e : \tau]\!] : \mathcal{D}[\![\tau]\!]]\!]$.

Now, using a limit construction analogous to §6.1.2 for obtaining a coupled *big-step* semantics $\langle \cdot, \cdot \rangle \colon P_\tau \to \mathcal{D}(V_\tau + \bot)$, where $V_\tau = \{(v : \tau, \mathcal{D}[\![v]\!] : \mathcal{D}[\![\tau]\!]) \mid v : \tau \text{ is a value}\}$, we prove a suitable continuity property of projection (see [51, Lemma F.55]) to conclude that Lemma 6.3 transfers to $\langle \cdot, \cdot \rangle$. Finally, observing that $V_{\mathbf{bool}} + \bot = \{(\text{true}, \text{true}), (\text{false}, \text{false}), \bot\}$ yields the desired claim:

Proof of Theorem 6.2. Let $v \in \{\text{true}, \text{false}, \bot\}$. We have

$$\begin{aligned}
& \langle e : \mathbf{bool} \rangle(v) \\
= \; & \pi_1(\langle e : \mathbf{bool}, \mathcal{D}[\![e : \mathbf{bool}]\!] : \mathbf{bool} \rangle)(v) && \text{(Lemma 6.3.1 for } \langle \cdot, \cdot \rangle) \\
= \; & \pi_1(\langle e : \mathbf{bool}, \mathcal{D}[\![e : \mathbf{bool}]\!] : \mathbf{bool} \rangle)(v) && (P_{\mathbf{bool}} \text{ is discrete}) \\
= \; & \langle e : \mathbf{bool}, \mathcal{D}[\![e : \mathbf{bool}]\!] : \mathbf{bool} \rangle((\text{true}, \text{true})) \cdot \delta_{\text{true}}(v) \\
& + \langle e : \mathbf{bool}, \mathcal{D}[\![e : \mathbf{bool}]\!] : \mathbf{bool} \rangle((\text{false}, \text{false})) \cdot \delta_{\text{false}}(v) \\
& + \langle e : \mathbf{bool}, \mathcal{D}[\![e : \mathbf{bool}]\!] : \mathbf{bool} \rangle(\bot) \cdot \delta_\bot(v) && \text{(definition of } \pi_1\text{, argument is discrete)} \\
= \; & \pi_2(\langle e : \mathbf{bool}, \mathcal{D}[\![e : \mathbf{bool}]\!] : \mathbf{bool} \rangle)(v) && \text{(definition of } \pi_2\text{, argument is discrete)} \\
= \; & \langle \mathcal{D}[\![e : \mathbf{bool}]\!] : \mathbf{bool} \rangle(v) \,. && \text{(Lemma 6.3.2 for } \langle \cdot, \cdot \rangle)
\end{aligned}$$

7 Implementation, Case Studies, and Evaluation

We have implemented the Slice type inference and discretization procedures in OCaml. To compute the interval masses of continuous distributions, we use the OCaml GSL package [18] for cumulative distribution function calculations. All experiments were run on an Apple M3 with 16 GB of RAM.

This section is driven by two research questions:

(1) Does Slice enable exact inference on programs combining continuous sampling, higher-order functions, and unbounded recursion?
(2) On programs that are within the scope of existing tools for exact inference, is the combination of Slice and discrete inference backends competitive?

Continuous Program

```
1  (fix resample x :=
2    let _ = observe(x >= 0.2) in
3    if x <= 0.5 then true
4    else if x <= 0.8 then resample x
5    else
6      let y = uniform(0,1) in
7        resample y) 1
```

Discretized Program

```
8  (fix resample x :=
9    let _ = observe(x >= 1) in
10   if x <= 1 then true
11   else if x <= 2 then resample x
12   else
13     let y = discrete(0.2,0.3,0.3,0.2) in
14       resample y) 3
```

Markov Chain Construction & Model Checking

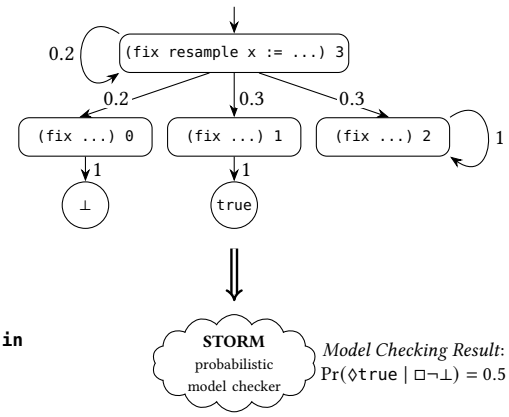


Fig. 15. Conditional resampling loop and its discretization. The continuous program is transformed into a finite-state discrete program (Slice alone takes less than 0.001s), which induces a finite Markov chain (generated in 0.001s) and can then be analyzed by Storm (with a model checking time of 0.024s). The ⊥-state in that Markov chain is entered when encountering an observation failure.

For the first question, we provide a collection of case studies that involve continuous distributions together with recursion, higher-order functions, lists, or conditioning in §7.1. For these case studies, we use Slice to transform the original programs to finite discrete models and analyze the resulting program with Storm [15] — a probabilistic model checker.

For the second question, we evaluate Slice+Dice and Slice+Roulette on two classes of benchmarks in Sections 7.2 and 7.3: (i) synthetic scaling benchmarks that let us vary program size and dependency structure in a controlled way, and (ii) benchmark suites from prior work [21, 40]. We compare against Sppl [40], an exact inference engine supporting continuous probabilistic programs, on benchmarks that *both* Slice and Sppl support, and we additionally report Psi [21] and Hakaru [37] timings for the representative prior-work benchmarks. We remark, however, that Sppl supports arithmetic, as well as (soft) conditioning on measure zero events, which Slice does not support. Our evaluation merely serves the purpose of investigating (i) the overhead of discretization and (ii) the benefits of being able to use efficient *BDD-based* inference back-ends (i.e., Dice and Roulette) for exact inference on *continuous programs*.

7.1 Slice+Storm: Case Studies

There is no exact inference system that, to our knowledge, can cover programs combining all three features: continuous sampling, higher-order functions, and unbounded recursion. As a consequence, there is no exact inference engine for the programs in Sections 2.6, 7.1.2 and 7.1.3. Instead, we use *probabilistic model checking*: The small-step semantics of our language (given in the extended version [51, Tables 1 and 2]) for discrete programs gives rise to a *Markov chain* modeling the stochastic process of program execution, inspired by [29] on imperative programs. If that Markov chain is finite, we can use the probabilistic model checker Storm [15] for exact inference. Let us start with an illustrative example.

7.1.1 Conditional Resampling Loop. Consider the continuous program depicted in Figure 15, containing continuous sampling, unbounded recursion, and conditioning inside of the recursion. Slice

automatically discretizes this program, also shown in Figure 15. We generate the corresponding Markov chain by unfolding the small-step semantics in a breadth-first manner. In the presence of unbounded recursion, this Markov chain *might* be infinite and this process might thus diverge. If, however, it terminates, the resulting Markov chain is finite and precisely captures the execution behavior of the discretized program. For this example, this is the case: Even though the discretized program contains unbounded recursion, its execution behavior can be modelled as a finite-state Markov chain[2], where $\bot$ is a special state entered when encountered an observation failure.

Now, using the probabilistic model checker Storm, we can perform exact inference: It is immediate that the Markov chain's probability $\Pr(\Diamond \texttt{true} \mid \Box\neg\bot)$ of eventually reaching true (in LTL: $\Diamond$true) *given* that we never enter the observation-failure state $\bot$ (in LTL: $\Box\neg\bot$) equals the measure of true in the program's normalized posterior distribution.

7.1.2 Random Interval Covering of $[0,1]$ *with Conditioning.* Our second case study extends the random interval covering example from §2.6 by conditioning inside of the recursion: We condition on *every* sample falling within *at most* one interval. The difference to the program in §2.6 is:

```
1  let non_overlap = fix non_overlap y :=
2    fun seen -> fun ys ->
3      match ys with
4      | nil -> true
5      | h :: t ->
6          let hit = (snd h) y in
7          if hit && seen then false
8          else non_overlap y (seen || hit) t
9      end in
10 let cover = fix cover st :=
11   if uniform(0,1) < 0.1 then st
12   else
13     let y = gaussian(0,1) in
14     let _ = observe(non_overlap y false st) in
15     cover (step y st)
16 in
17 let final = cover init in
18 is_covered final
```

The key new component is the recursive function `non_overlap` (l. 1), which checks whether a sampled point y hits more than one interval of the initially sampled intervals. The loop `cover` (l. 10) repeatedly samples a Gaussian point and immediately conditions on this predicate via **`observe`** (l. 14), so only executions in which the point lies in at most one interval are retained. The measure of true in the conditional posterior distribution is thus the probability that this unbounded process hits every interval at least once *given* that each sampled point falls in at most one interval.

The program remains discretizable: each sampled point is still used only through comparisons against finitely many sampled interval boundaries, and `non_overlap` depends only on the resulting hit pattern, not on the exact real value of y. Slice replaces each Gaussian draw by its interval index, yielding a discrete program (shown in [51, Figure 22]) with a finite exact state space. Slice takes 0.001s, and we generate a Markov chain consisting of 268878 states in 80 seconds. Storm takes 4.7 seconds for model checking, and reports that the sought-after probability is 0.000227....

Currently, our Markov chain construction process is rather naive: extending the number of sampling points or intervals yields the Markov chain to be infeasibly large. Developing efficient symbolic discrete inference engines for higher-order programs with unbounded recursion is not in the scope of this paper. However, our approach is an excellent motivation for this future direction.

7.1.3 Bloom Filter. We model a probabilistically populated variant of a Bloom filter [7], which is a data structure using hash functions for storing data space efficiently: one can insert data into a Bloom filter and subsequently query whether a given data point is stored in it. Due to aggressive compression via hashing, this process can be faulty: while it can never produce false-negatives, it can produce false-positives. Now conisder the following program (full program in [51, Figure 23]):

[2]We apply a straightforward optimization: Intermediate execution steps that do not branch probabilistically can be collapsed into a single transition. This is the reason why all states in Figure 15 correspond to the header of the recursive function.

```
1  let insert = fun q -> fun h1 ->
2      fun h2 -> fun bf ->
3    let index1 = h1 q in
4    let index2 = h2 q in
5    let bf1 = set_true bf index1 in
6    set_true bf1 index2 in
7
8  let search = fun q -> fun h1 ->
9      fun h2 -> fun bf ->
10   let index1 = h1 q in
11   let index2 = h2 q in
12   (get bf index1) && (get bf index2) in
13
14 let h1 = fun q ->
15   if q <= -2 then 0
16   else if q <= 0 then 1
17   else if q <= 2 then 2
18   else 3 in
19 let h2 = fun q ->
20   if q <= -2 then 3
21   else if q <= 0 then 2
22   else if q <= 2 then 1
23   else 0 in
24
25 let init = (false, (false, (false, false))) in
26 let key = gaussian(0,1) in
27
28 let search_before_full = fix loop bf :=
29   if (full bf) then bf
30   else if (search key h1 h2 bf) then bf
31   else
32     let q = gaussian(0,1) in
33     loop (insert q h1 h2 bf) in
34
35 let bf_final = (search_before_full init) in
36   not (full bf_final)
```

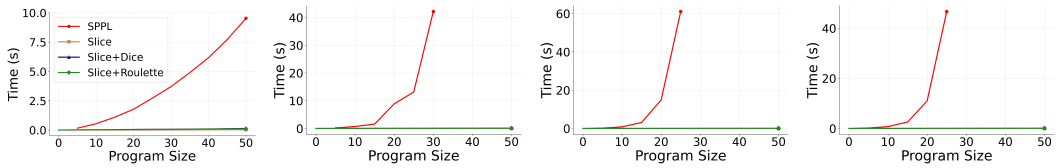
The program samples a Gaussian key (l. 26) and starts from an empty four-bit Bloom filter (l. 25). It then builds the filter recursively by drawing fresh Gaussian samples (l. 32) and inserting each one into the filter by hashing the sampled real twice using the comparison-based hash functions `h1` and `h2` (l. 14 and 19), following a locality-sensitive hashing technique that hashes similar input items into the same buckets. Each insertion sets the corresponding bits in the filter using `insert` (l. 1). The unbounded recursion continues until either the filter becomes full or the key is reported present by `search` (l. 8). The final Boolean result, `not (full bf_final)` (l. 36), therefore asks whether the key was found strictly before the filter became full.

The key simplification is that the hash functions never inspect the exact value of a sample. They only ask which region the sample falls into, using the cut points -2, 0, and 2. These cuts partition the real line into four regions, which already coincide with the Bloom filter's index space. Slice therefore replaces each Gaussian draw by its bucket index while preserving exactly the probability of every collision pattern relevant to insertion and search. The fully discretized program is shown in the extended version [51, Figure 24]. Slice takes less than 0.001s, and we generate a Markov chain consisting of 14 states in 0.023 seconds. Storm takes 0.001 seconds for model checking, and reports the probability 0.91314....

7.2 Slice+Dice and Slice+Roulette: New Benchmarks for Scaling

In the absence of unbounded recursion, we can use Dice or Roulette as inference backends. We designed scalable benchmarks of conditionally independent continuous programs with varying variable dependencies. Formally, all programs in our benchmark can be described by the structure: $X_i \perp\!\!\!\perp \{ X_j : j < i, j \neq \pi(i) \} \mid X_{\pi(i)}\}$, where $\pi(i) \in \{1, \ldots, i-1\}$ is the parent index of X_i. Each variable X_i depends directly on only one earlier variable, namely its parent $X_{\pi(i)}$; conditioned on $X_{\pi(i)}$, it is independent of all other preceding variables. Detailed listings and graphs for variants of these benchmarks appear in the extended version [51, § G.1.1]. Results are depicted in Figure 16.

We report runtime (seconds) for (i) Sppl, (ii) Slice (discretization only), (iii) Slice+Dice (discretization plus inference via Dice), and (iv) Slice+Roulette (discretization plus inference via Roulette). The timeout is 120 seconds. Program sizes are evaluated in increments of 5, from 0 to 50. For each size, we generate 10 random instances and report average runtime. We observe that discretization remains small across sizes, and takes less than 0.01 seconds for each program. End-to-end runtimes for Slice with discrete backends grow more mildly on these families; Sppl



(a) Cond. Ind. – Fork (b) Cond. Ind. – Rand. (c) Alt. Guard (d) Alt. Guard – Rand.

Fig. 16. Scaling behavior on representative benchmark families (additional families are provided in the extended version [51, § G.1.1]). The plots emphasize asymptotic growth-rate differences and the resulting timeout frontier as program size increases. Time-out=120s; a disappearing line indicates timeout.

reaches a timeout frontier at larger instances. We conclude that using efficient BDD-based back-ends for exact inference on continuous programs via discretization can pay off for programs whose control flow induces strong conditional independence structure, allowing the resulting discrete models to remain compact even as program size grows.

7.3 Slice+Dice and Slice+Roulette: Existing Benchmarks

We next use benchmark suites from prior work, detailed in the extended version [51, § G, Tables 5–7].

The decision tree benchmarks from [1] assess fairness properties under different population models. In the extended version [51, Tables 5 and 6], DT_n denotes a tree with n conditionals, the second column specifies the population model, and the remaining columns report runtimes. We split Sppl execution into translation and fairness-judgment phases to make phase costs explicit. We additionally report runtimes for Psi and Hakaru. For Hakaru, we separate Disintegrate, Simplify, and Total runtimes. Hakaru's disintegration phase transforms the original probabilistic program into a new Hakaru program representing a conditional distribution, which is then simplified by Maple to perform exact inference through computer algebra. Thus, the Total column reports the timing for the full exact inference pipeline. We observe that discretization itself (Slice) becomes expensive for larger n, eventually yielding a timeout. Regarding end-to-end runtimes, Slice+Dice mostly outperforms Sppl, whereas Slice+Roulette is mostly inferior to Sppl for larger n.

The extended version [51, Table 7] reports on benchmarks drawn from the Psi and Sppl benchmark sets. For benchmarks adapted by Sppl from Psi, we use the modified versions provided in the Sppl Github repo, which replace certain continuous choices with finite ones. Translating these versions to Slice enables a direct comparison between Slice and Sppl. For completeness, we also report Psi/Hakaru results on the original, unmodified programs, which remain outside the class supported by both Slice and Sppl. Moreover, programs requiring numeric transformations or arithmetic *remain outside the language features of* Slice. Like in the decision tree benchmarks, we report runtimes for (i) Slice+Dice, (ii) Slice+Roulette, (iii) Sppl, (iv) Psi, (v) Hakaru (Disintegrate), and (v) Hakaru (Simplify) in seconds. We observe that Slice+Dice and Slice+Roulette are competitive with Sppl, Psi, and Hakaru (Disintegrate + Simplify).

8 Related and Future Work

We start with related work and conclude with future work.

Discretization approaches. Holtzen et al. [25] are the closest prior work on exact discretization. They develop a probabilistic predicate abstraction framework that, given predicates of interest, constructs an abstract probabilistic program and proves soundness for events expressible in the predicate domain. For a first-order, loop-free fragment of our language, Slice can be viewed similarly: inferred cuts such as $x < c$ and $x \leq c$ induce predicates over which the program is discretized.

The key difference is that Slice *automatically infers and propagates* the relevant cut/value sets through higher-order functions, recursion, and observations, then compiles to a discrete program for existing exact backends. By contrast, Holtzen et al. [25] assume predicates are supplied a priori, explicitly identify useful predicate discovery as a hard problem, and restrict their semantics to measurable functions, excluding arbitrary unbounded loops and higher-order functions. Beyond this, other discretization methods use approximate discretization [11, 19, 27]. Recent work also uses approximate discretization to compute posterior bounds for recursive higher-order programs [4]. Vice versa, Leios [32] approximates a fully continuous program from a discrete one.

Discrete-only probabilistic languages. Roulette [35] extends Racket with first-class support for finitely-supported distributions and leverages symbolic evaluation to compile queries into weighted model-counting problems, enabling scalable exact inference on discrete programs. Dice [26] follows a similar philosophy in an OCaml DSL, compiling discrete programs to weighted model counting. Pluck [8] enables exact and approximate inference in discrete probabilistic programs based on lazy evaluation. Earlier work in probabilistic logic programming, most prominently ProbLog [14], annotates Prolog facts with probabilities and reduces inference to weighted Boolean formulas. PERPL is a system that performs exact inference on recursive calls over recursive data [10]. Caesar [42] is a quantitative program verification infrastructure for discrete probabilistic programs that together with the approach from [2] enables the verification of continuous programs.

Exact inference on continuous programs. Many existing systems perform exact inference directly in continuous/hybrid languages. Psi [21] and Hakaru [37] support symbolic exact inference capabilities with different algebraic strengths, though symbolic generality can make some tractable instances challenging. Symbo [17] leverages weighted model integration [3] for exact inference in hybrid domains, using sentinel decision diagrams as the representation and the PSI solver to symbolically integrate over continuous variables. SPPL [40] emphasizes efficient exact symbolic inference for its fragment and supports features we do not currently automate, notably numeric transformations and conditioning on measure-zero events. λPSI [22] supports exact inference for higher-order continuous programs with nested inference and bounded recursion. Generating-function systems such as Genfer [53], Geni [33], and Prodigy [31] collectively support exact inference for infinite-support models and restricted unbounded-control models, but not general higher-order programs. PERPL [10] supports higher-order functions and unbounded recursion, but not continuous random variables. GuBPI [4] supports recursive higher-order programs with continuous distributions, but computes guaranteed converging bounds rather than exact posteriors. Slice targets the combination needed for our case studies: exact inference for programs that combine continuous sampling, higher-order functions, and unbounded recursion.

Relational proof techniques. Our soundness argument uses a coupling-style semantic relation between source and discretized executions. This connects directly to logical-relations and relational-semantics techniques for probabilistic programs [6, 12, 48], adapted here to a global, non-local, type-guided transformation. Notably, [6] considers only discrete probabilistic programs without conditioning, while [12, 48] support continuous distributions and scoring. More recently, [54] considered probabilistic programs with nested queries and recursion using a step-indexed model.

Future Work. We would like to extend our work to support tractable symbolic integration for richer classes of continuous distributions (i.e. conjugate priors or other distributions admitting closed-form expressions), thus broadening the class of programs that Slice can discretize exactly. We also plan to investigate symbolic techniques à la Wang et al. [49] to speed-up inference in the presence of recursion.

Data-Availability Statement

A reproducible artifact for the evaluation in §7 is available on Zenodo [50]. The extended version of this paper (e.g. with appendices) is also available [51].

Acknowledgments

We are grateful to our OOPSLA reviewers who helped us improve the paper. We also thank Cheng Zhang for providing feedback on an initial draft of our paper, as well as the Silva Lab retreat guests for helpful discussions. This work was supported by the NSF Graduate Research Fellowship Program under Grant No. DGE-2139899. Additionally, this research was developed with funding from the Defense Advanced Research Projects Agency (DARPA) and the Office of Naval Research (ONR). The views, opinions, and/or findings contained in this material are those of the authors and should not be interpreted as representing the official views or policies of the Department of the Navy, DARPA, or the U.S. Government.

References

[1] Aws Albarghouthi, Loris D'Antoni, Samuel Drews, and Aditya V. Nori. 2017. FairSquare: Probabilistic Verification of Program Fairness. In *Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*. doi:10.1145/3133904

[2] Kevin Batz, Joost-Pieter Katoen, Francesca Randone, and Tobias Winkler. 2025. Foundations for Deductive Verification of Continuous Probabilistic Programs: From Lebesgue to Riemann and Back. In *Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*. doi:10.1145/3720429

[3] Vaishak Belle, Andrea Passerini, and Guy Van den Broeck. 2015. Probabilistic Inference in Hybrid Domains by Weighted Model Integration. In *International Joint Conference on Artificial Intelligence (IJCAI)*. https://dl.acm.org/doi/10.5555/2832581.2832636

[4] Raven Beutner, C.-H. Luke Ong, and Fabian Zaiser. 2022. Guaranteed Bounds for Posterior Inference in Universal Probabilistic Programming. In *Programming Language Design and Implementation (PLDI)*. doi:10.1145/3519939.3523721

[5] 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. *JMLR* (2019).

[6] Aleš Bizjak and Lars Birkedal. 2015. Step-Indexed Logical Relations for Probability. In *Foundations of Software Science and Computation Structures (FoSSaCS)*. doi:10.1007/978-3-662-46678-0_18

[7] Burton H. Bloom. 1970. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM* 13, 7 (1970), 422–426. doi:10.1145/362686.362692

[8] Maddy Bowers, Alexander K. Lew, Joshua B. Tenenbaum, Armando Solar-Lezama, and Vikash K. Mansinghka. 2025. Stochastic Lazy Knowledge Compilation for Inference in Discrete Probabilistic Programs. In *Programming Language Design and Implementation (PLDI)*. doi:10.1145/3729325

[9] 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. *J. Stat. Softw.* (2017). doi:10.18637/jss.v076.i01

[10] David Chiang, Colin McDonald, and Chung-chieh Shan. 2023. Exact Recursive Probabilistic Programming. In *Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*. doi:10.1145/3586050

[11] Guillaume Claret, Sriram K. Rajamani, Aditya V. Nori, Andrew D. Gordon, and Johannes Borgström. 2013. Bayesian Inference Using Data Flow Analysis. In *Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. doi:10.1145/2491411.2491423

[12] Ryan Culpepper and Andrew Cobb. 2017. Contextual Equivalence for Probabilistic Programs with Continuous Random Variables and Scoring. In *European Symposium on Programming (ESOP)*. doi:10.1007/978-3-662-54434-1_14

[13] 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 *Programming Language Design and Implementation (PLDI)*. doi:10.1145/3314221.3314642

[14] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. 2007. ProbLog: A Probabilistic Prolog and Its Application in Link Discovery. In *International Joint Conference on Artificial Intelligence (IJCAI)*. https://dl.acm.org/doi/10.5555/1625275.1625673

[15] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. 2017. A Storm is Coming: A Modern Probabilistic Model Checker. In *Computer Aided Verification (CAV)*. doi:10.1007/978-3-319-63390-9_31

[16] Joshua V. Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alexander A. Alemi, Matthew D. Hoffman, and Rif A. Saurous. 2017. TensorFlow Distributions. arXiv:1711.10604 doi:10.48550/arXiv.1711.10604

[17] Pedro Zuidberg Dos Martires, Anton Dries, and Luc De Raedt. 2019. Exact and Approximate Weighted Model Integration with Probability Density Functions Using Knowledge Compilation. In *AAAI Conference on Artificial Intelligence (AAAI)*. doi:10.1609/aaai.v33i01.33017825

[18] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi. 2009. *GNU Scientific Library Reference Manual* (3rd ed.). http://www.gnu.org/software/gsl/

[19] Poorva Garg, Steven Holtzen, Guy Van den Broeck, and Todd Millstein. 2024. Bit Blasting Probabilistic Programs. In *Programming Language Design and Implementation (PLDI)*. doi:10.1145/3656412

[20] Hong Ge, Kai Xu, and Zoubin Ghahramani. 2018. Turing: A Language for Flexible Probabilistic Inference. In *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics (AISTATS) (Proceedings of Machine Learning Research, Vol. 84)*. PMLR, 1682–1690. https://proceedings.mlr.press/v84/ge18b.html

[21] Timon Gehr, Sasa Misailovic, and Martin Vechev. 2016. PSI: Exact Symbolic Inference for Probabilistic Programs. In *Computer Aided Verification (CAV)*. doi:10.1007/978-3-319-41528-4_4

[22] Timon Gehr, Samuel Steffen, and Martin Vechev. 2020. λPSI: Exact Inference for Higher-Order Probabilistic Programs. In *Programming Language Design and Implementation (PLDI)*. doi:10.1145/3385412.3386006

[23] Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. 2008. Church: A Language for Generative Models. In *Conference on Uncertainty in Artificial Intelligence (UAI)*. https://dl.acm.org/doi/10.5555/3023476.3023503

[24] Noah D. Goodman and Andreas Stuhlmüller. 2014. The Design and Implementation of Probabilistic Programming Languages. http://dippl.org

[25] Steven Holtzen, Guy Van den Broeck, and Todd Millstein. 2018. Sound Abstraction and Decomposition of Probabilistic Programs. In *Proceedings of the 35th International Conference on Machine Learning (ICML) (Proceedings of Machine Learning Research, Vol. 80)*. PMLR, 1999–2008. https://proceedings.mlr.press/v80/holtzen18a.html

[26] Steven Holtzen, Guy Van den Broeck, and Todd Millstein. 2020. Scaling Exact Inference for Discrete Probabilistic Programs. In *Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*. doi:10.1145/3428208

[27] Zixin Huang, Saikat Dutta, and Sasa Misailovic. 2021. AQUA: Automated Quantized Inference for Probabilistic Programs. In *International Symposium on Automated Technology for Verification and Analysis (ATVA)*. doi:10.1007/978-3-030-88885-5_16

[28] Jules Jacobs. 2021. Paradoxes of Probabilistic Programming: And How to Condition on Events of Measure Zero with Infinitesimal Probabilities. In *Principles of Programming Languages (POPL)*. doi:10.1145/3434339

[29] Nils Jansen, Christian Dehnert, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Lukas Westhofen. 2016. Bounded Model Checking for Probabilistic Programs. In *International Symposium on Automated Technology for Verification and Analysis (ATVA)*. doi:10.1007/978-3-319-46520-3_5

[30] Olav Kallenberg. 1997. *Foundations of Modern Probability*. Springer-Verlag, New York.

[31] Lutz Klinkenberg, Christian Blumenthal, Mingshuai Chen, Darion Haase, and Joost-Pieter Katoen. 2024. Exact Bayesian Inference for Loopy Probabilistic Programs Using Generating Functions. In *Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*. doi:10.1145/3649844

[32] Jacob Laurel and Sasa Misailovic. 2020. Continualization of Probabilistic Programs With Correction. In *European Symposium on Programming (ESOP)*. doi:10.1007/978-3-030-44914-8_14

[33] Jianlin Li and Yizhou Zhang. 2025. Compiling with Generating Functions. In *International Conference on Functional Programming (ICFP)*. doi:10.1145/3747534

[34] Tom Minka, John M. Winn, John Guiver, Yordan Zaykov, David Fabian, and Jim Bronskill. 2018. Infer.NET 0.3. http://dotnet.github.io/infer

[35] Cameron Moy, Jack Czenszak, John M. Li, Brianna Marshall, and Steven Holtzen. 2025. Roulette: A Language for Expressive, Exact, and Efficient Discrete Probabilistic Programming. In *Programming Language Design and Implementation (PLDI)*. doi:10.1145/3729334

[36] James R. Munkres. 2000. *Topology* (2nd ed.). Prentice Hall, Upper Saddle River, NJ.

[37] Praveen Narayanan, Jacques Carette, Wren Romano, Chung chieh Shan, and Robert Zinkov. 2016. Probabilistic Inference by Program Transformation in Hakaru (System Description). In *International Symposium on Functional and Logic Programming (FLOPS)*. doi:10.1007/978-3-319-29604-3_5

[38] Martin Odersky, Martin Sulzmann, and Martin Wehr. 1999. Type Inference with Constrained Types. *Theory and Practice of Object Systems* (1999). doi:10.1002/(SICI)1096-9942(199901/03)5:1<35::AID-TAPO4>3.0.CO;2-4

[39] Avi Pfeffer. 2009. *Figaro: An Object-Oriented Probabilistic Programming Language*. Technical Report. Charles River Analytics. https://www.cs.tufts.edu/comp/150FP/archive/avi-pfeffer/figaro.pdf

[40] Feras A. Saad, Martin C. Rinard, and Vikash K. Mansinghka. 2021. SPPL: Probabilistic Programming with Fast Exact Symbolic Inference. In *Programming Language Design and Implementation (PLDI)*. doi:10.1145/3453483.3454078

[41] John Salvatier, Thomas V. Wiecki, and Christopher Fonnesbeck. 2016. Probabilistic Programming in Python Using PyMC3. *PeerJ CS* (2016). doi:10.7717/peerj-cs.55

[42] Philipp Schroer, Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2023. A Deductive Verification Infrastructure for Probabilistic Programs (Extended Version). In *Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*. doi:10.1145/3622870

[43] Elias M. Stein and Rami Shakarchi. 2005. *Real Analysis: Measure Theory, Integration, and Hilbert Spaces* (1st ed.). Princeton Lectures in Analysis, Vol. 3. Princeton University Press, Princeton, NJ.

[44] Terence Tao. 2011. *An Introduction to Measure Theory* (1st ed.). Graduate Studies in Mathematics, Vol. 126. American Mathematical Society, Providence, RI.

[45] Nazanin Tehrani, Nimar S. Arora, Yucen Lily Li, Kinjal Divesh Shah, David Noursi, Michael Tingley, Narjes Torabi, Sepehr Masouleh, Eric Lippert, and Erik Meijer. 2020. Bean Machine: A Declarative Probabilistic Programming Language For Efficient Programmable Inference. In *Proceedings of the 10th International Conference on Probabilistic Graphical Models (PGM) (Proceedings of Machine Learning Research, Vol. 138)*. PMLR, 485–496. https://proceedings.mlr.press/v138/tehrani20a.html

[46] David Tolpin, Jan-Willem van de Meent, Hongseok Yang, and Frank Wood. 2016. Design and Implementation of Probabilistic Programming Language Anglican. In *Implementation and Application of Functional Languages (IFL)*. doi:10.1145/3064899.3064910

[47] Dustin Tran, Alp Kucukelbir, Adji B. Dieng, Maja Rudolph, Dawen Liang, and David M. Blei. 2016. Edward: A Library for Probabilistic Modeling, Inference, and Criticism. arXiv:1610.09787 doi:10.48550/arXiv.1610.09787

[48] Mitchell Wand, Theophilos Giannakopoulos, Andrew Cobb, and Ryan Culpepper. 2018. Contextual Equivalence for a Probabilistic Language with Continuous Random Variables and Recursion. In *International Conference on Functional Programming (ICFP)*. doi:10.1145/3236782

[49] Peixin Wang, Tengshun Yang, Hongfei Fu, Guanyan Li, and C.-H. Luke Ong. 2024. Static Posterior Inference of Bayesian Probabilistic Programming via Polynomial Solving. In *Programming Language Design and Implementation (PLDI)*. doi:10.1145/3656432

[50] Katherine Wu, Jules Jacobs, Kevin Batz, and Alexandra Silva. 2026. Artifact: Type-Directed Discretization of Probabilistic Programs. Zenodo. doi:10.5281/zenodo.21780754

[51] Katherine Wu, Jules Jacobs, Kevin Batz, and Alexandra Silva. 2026. Type-Directed Discretization of Probabilistic Programs (Extended Version). arXiv:2608.16093 doi:10.48550/arXiv.2608.16093

[52] 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 (ICML) (Proceedings of Machine Learning Research, Vol. 80)*. PMLR, 5343–5352. https://proceedings.mlr.press/v80/wu18f.html

[53] Fabian Zaiser, Andrzej S. Murawski, and Luke Ong. 2023. Exact Bayesian Inference on Discrete Models via Probability Generating Functions: A Probabilistic Programming Approach. In *Conference on Neural Information Processing Systems (NeurIPS)*. doi:10.52202/075280-0113

[54] Yizhou Zhang and Nada Amin. 2022. Reasoning about "Reasoning about Reasoning": Semantics and Contextual Equivalence for Probabilistic Programs with Nested Queries and Recursion. In *Principles of Programming Languages (POPL)*. doi:10.1145/3498677

Contents

A Slice Examples (Continued)

This section is a continuation of §2, describing more in detail the following examples: Sections 2.3 to 2.6. Additionally, we include an extra example based on the Indian GPA problem, a canonical example in the probabilistic programming literature [52].

A.1 Control Flow

We revisit the control-flow example from §2.3:

```
let x = if flip() then uniform(0,2) else gaussian(0,1) in
let y = if 1.5 > x then 1.8 else 0.3 in x <= y
```

The main feature of this program is that the comparison $x \leq y$ is governed by control flow: the right-hand side is not a single constant, but may evaluate to either 0.3 or 1.8 depending on the outcome of the earlier test $1.5 > x$. Consequently, the discretization of x must account for both these values.

We show, at a high-level, how the float type information for this program is propagated. First, Figure 17 makes these dependencies explicit using a constraint graph over float-typed subexpressions. Solid edges denote data flow through conditionals, while dashed edges denote comparison

```
let x = if flip() then uniform(0,2) else gaussian(0,1) in
let y = if 1.5 > x then 1.8 else 0.3 in x <= y
```

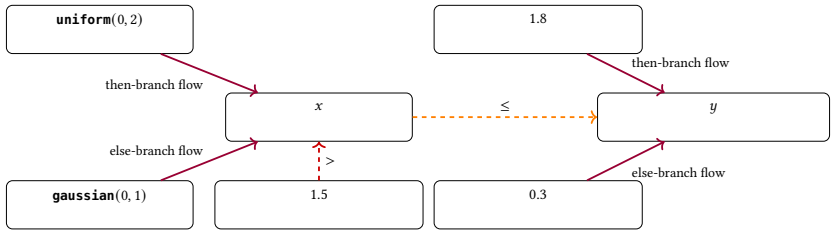


Fig. 17. Constraint graph for float types. Solid arrows represent data flow, and dashed arrows constraints.

constraints. In this example, the comparisons $1.5 > x$ and $x \leq y$ jointly determine the cut structure relevant to x. The data flow edges record how continuous samples from **uniform**$(0, 2)$ and **gaussian**$(0, 1)$ flow into x, while the constants 1.8 and 0.3 flow into y through its two branches.

The propagation process of float type information proceeds in four stages (Figure 18). First, literal constants contribute singleton value sets, while continuous samples are assigned value set $\top$ (Figure 18a). Second, value sets propagate forward along data-flow edges, yielding in particular that y has value set $\{0.3, 1.8\}$ (Figure 18b). Third, comparison sites generate cut information: $1.5 > x$ contributes the threshold 1.5, and $x \leq y$ contributes the possible values of y, namely 0.3 and 1.8 (Figure 18c). Since both operands of a comparison must be discretized against compatible interval boundaries, these cut sets are unified across the compared expressions. Finally, the resulting cut set propagates backward along data flow edges to all contributing subexpressions (Figure 18d).

The final cut set is

$$\{\leq 0.3, <1.5, \leq 1.8\},$$

inducing the intervals $(-\infty, 0.3]$, $(0.3, 1.5)$, $[1.5, 1.8]$, and $(1.8, \infty)$.

A.2 Discrete Latent Variable

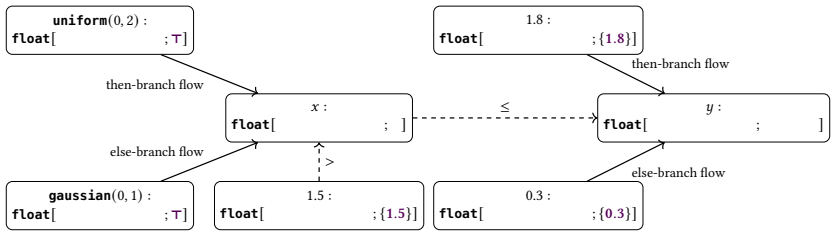
We revisit the discrete latent variable example from §2.4:

```
let x = if flip() then 0.5 else 1.5 in gaussian(0, x) < 0.5
```

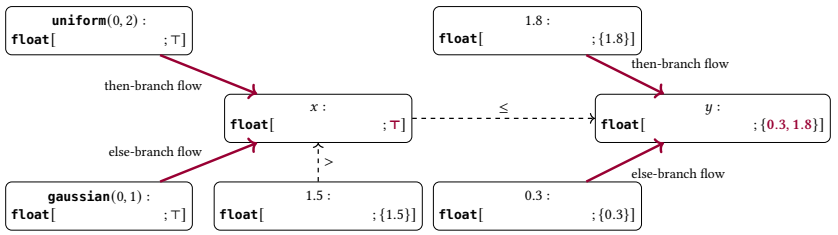
The essential feature here is that the parameter of the continuous distribution is itself determined by a discrete choice. Thus, the discretization of the program must also account for the fact that the Gaussian is instantiated at two distinct parameter values.

Figure 19 presents the corresponding constraint graph. The variable x is formed by control flow from the two constants 0.5 and 1.5, and therefore has value set $\{0.5, 1.5\}$. Because x appears as a parameter to **gaussian**$(0, x)$, Slice must recover which concrete value of x was selected in order to instantiate the appropriate Gaussian distribution. The figure records this dependency by the *recovers* constraint on x.

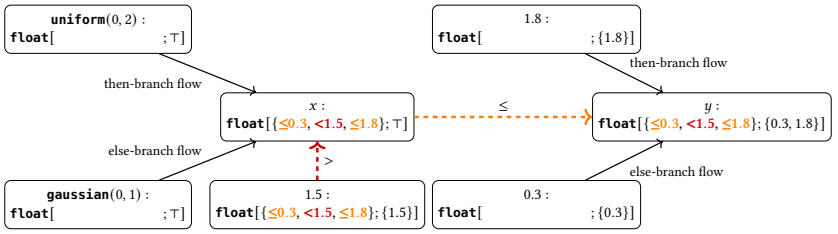
Operationally, the recovers constraint requires that the values in the value set of x be represented by distinct discrete indices. This is why the type system uses non-strict cuts for such values: the cut set $\{\leq 0.5, \leq 1.5\}$ induces a partition in which 0.5 and 1.5 occupy distinguishable regions, allowing them to be mapped to separate cases in the discretized program. The final comparison **gaussian**$(0, x) < 0.5$ contributes the cut <0.5 for the sampled Gaussian result, while the value-set information on x determines which discretized Gaussian is selected.



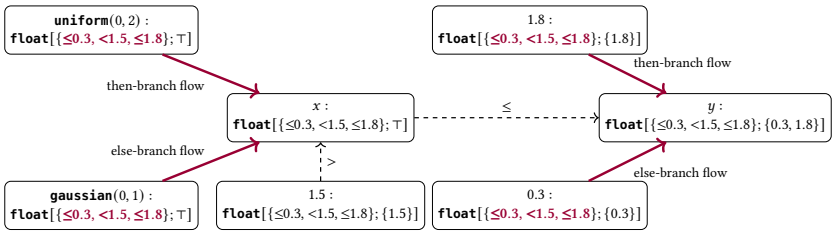
(a) *Initialize value sets.* Literal floats are annotated with their numeric value, and continuous distributions with $\top$.



(b) *Propagate value sets (forward along data flow edges).* Any expression formed by an if-expression inherits the union of the value sets of its branches.



(c) *Generate cut sets.* Comparisons contribute threshold values (e.g., 1.5, 0.3, 1.8) to the cut sets of participating expressions. Cut sets of expressions on both sides of a comparison (e.g. $1.5 > x$ and $x \leq y$) are unified to ensure consistent discretization.



(d) *Propagate cut sets (backward along data flow edges).* Cuts introduced at comparison sites are pushed backward along the edges in Figure 18b so that all contributing subexpressions share the same interval boundaries for discretization.

Fig. 18. Constraint graphs showing value-set flow and cut-set propagation over float-typed subexpressions for example §2.3. Colored elements indicate the components of the float types activated at each step.

```
let x = if flip() then 0.5 else 1.5 in gaussian(0, x) < 0.5
```

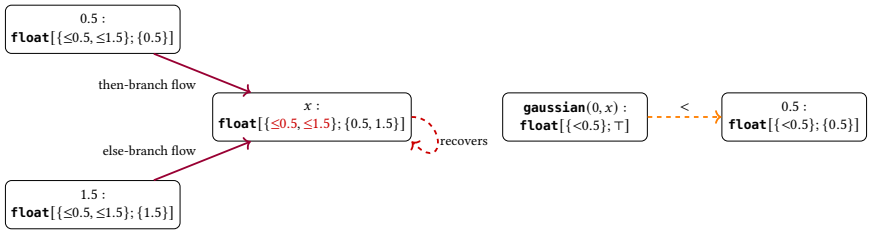


Fig. 19. Constraint graph populated with float type information. As before, solid arrows represent data flow, and dashed arrows represent comparison constraints. Using x as a Gaussian parameter induces a *recovers* constraint, the red dashed self-loop, which ensures a unique discrete index for each value of x.

A.3 Indian GPA Problem

This example involves both discrete and continuous random variables and models a student's GPA based on nationality (India or USA) and whether they have a perfect record. The GPA is deterministic for perfect students (10.0 for India, 4.0 for USA) but drawn from a uniform distribution otherwise. Slice correctly handles conditioning on a positive-measure atom at 10:

```
let nationality = discrete(0.5, 0.5) in
let perfect = discrete(0.01, 0.99) in
let gpa = if nationality <= 0 then
        if perfect <= 0 then
          (10.0 : float[{<10,<=10}; {10}])
        else
          (uniform(0, 10) : float[{<10,<=10}; T])
      else
        if perfect <= 0 then
          (4.0 : float[{<10,<=10}; {4}])
        else
          (uniform(0, 4) : float[{<10,<=10}; T])
      : float[{<10,<=10}; T] in
let _ = observe(gpa : float[{<10,<=10}; T] <= 10 : float[{<10,<=10}; {10}]
          && gpa : float[{<10,<=10}; T] >= 10 : float[{<10,<=10}; {10}]) in
nationality > 0
```

The type system propagates the cuts <10 and ≤ 10 for all GPA expressions. These cuts induce the three regions:

$$(-\infty, 10), \qquad \{10\}, \qquad (10, \infty).$$

Under this partition, both continuous branches, `uniform(0,10)` and `uniform(0,4)`, place all of their probability mass in the first region. The constant 10.0 is mapped to the singleton region $\{10\}$, while the constant 4.0 is mapped to the region $(-\infty, 10)$. The resulting discretized program is therefore:

```
let nationality = discrete(0.5, 0.5) in
let perfect = discrete(0.01, 0.99) in
let gpa = if nationality <= 0 then
        if perfect <= 0 then 1
        else
          discrete(1.0, 0.0, 0.0)
```

```
      else
        if perfect <= 0 then 0
        else
          discrete(1.0, 0.0, 0.0) in
let _ = observe(gpa <= 1 && gpa >= 1)
nationality > 0
```

After discretization, the equality test `gpa = 10` becomes the discrete test that the GPA index is exactly the one corresponding to the singleton region $\{10\}$.

A.4 Point and Measure-Zero Observations

Slice can condition on an exact value when that event has strictly positive probability mass, as in the Indian GPA example in §A.3. In that case, conditioning is well-defined and Slice returns the correct answer. The exact set of programs supported by Slice is therefore somewhat larger than one would initially expect, while not covering full point observations on continuous samples.

The mechanism is the same as in §A.3: the cuts $<c$ and $\leq c$ induce the partition

$$(-\infty, c), \qquad \{c\}, \qquad (c, \infty),$$

so any probability mass assigned exactly to c is tracked as its own discrete outcome.

More precisely:

(1) Slice cannot handle programs where every control-flow path conditions on an event of measure zero, and therefore programs with exact observations on continuous samples are in general out of scope, i.e. the programs one typically writes in e.g. Stan are out of scope unless one widens exact conditioning to a small but finite interval.
(2) Slice can handle programs where at least one control-flow path conditions only on strictly-positive-probability events, even if that is a point observation, and even if other paths condition only on measure-zero events.
(3) It does not matter whether an event is measure-zero because it is point-conditioning on a continuous sample, or whether it is measure-zero for another reason, e.g. **observe**(**uniform**$(0,1) > 3$) or simply **observe**(false). The critical distinction for **observe**(E) is whether $\Pr(E) = 0$, not whether E is of the form "$x = 3.2$".
(4) Slice will never give the wrong answer, even when all paths condition on measure-zero events. It will give the empty subdistribution, indicating that under a rejection-sampling interpretation, all samples are rejected with probability 1. Thus the user is notified, and Slice never silently gives a wrong answer.

For instance, this program:

```
let x = uniform(0,1) in
if flip() then
  let _ = observe(x >= 0.5 && x <= 0.5) in true
else
  let _ = observe(x <= 0.3 && x >= 0.3) in false
```

results in:

$$\Pr(\text{true}) = 0 \qquad \Pr(\text{false}) = 0.$$

Whereas this program:

```
let x = uniform(0,1) in
if flip() then
  let _ = observe(x >= 0.5 && x <= 0.5) in true
```

```
(* Returns true if y is in [lo,hi) *)
let within = fun lo -> fun hi -> fun y ->
  (y >= lo) && (y < hi) in

(* Function for sampling interval bounds *)
let b = fun _ ->
  discrete(0.18: 0.05, 0.11: 0.24,
           0.21: 0.37, 0.09: 0.58,
           0.41: 0.89) in

(* Initialize intervals by sampling their
respective (possibly overlapping) bounds *)
let init =
  (false, within 1 (b ())) ::
  (false, within (b ()) (b ())) ::
  (false, within (b ()) (b ())) ::
  (false, within (b ()) 8) ::
  nil in

(* Update flags depending on y falling
   within the respective interval *)
let update = fix update y := fun ys ->
  match ys with
    | nil -> nil
    | h :: t ->
        ((fst h || ((snd h) y)), snd h)
        :: (update y t)
  end in

(* Sample from a discretized gaussian,
    then use the resulting value to update
    flags *)
let step = fun ys ->
   update (discrete(0.5, 0.0199388,
                    0.0748961, 0.0494739,
                    0.0747339, 0.0632619,
                    0.0309625, 0.0280777,
                    0.158655)) ys in

 (* Number of samples determined by
    unbounded recursion *)
 let cover = fix cover st :=
   if discrete(0.1, 0.9) < 1 then st
   else cover (step st) in

(* Have all intervals been covered at
   least once? *)
let is_covered = fix is_covered ys :=
  match ys with
    | nil -> true
    | h :: t -> (fst h) && (is_covered t)
  end in

(* Run entire process on sampled intervals *)
let final = cover init in
  is_covered final
```

Fig. 20. A *discretized* infinite stochastic process for random interval covering over $[0, 1]$, involving continuous sampling, higher-order recursion, and lists.

```
  else
    let _ = observe(x <= 0.3001 && x >= 0.3) in false
```

results in:

$$\Pr(\text{true}) = 0 \qquad \Pr(\text{false}) = 1.$$

This is the correct behavior: consider repeatedly running the program under a rejection-sampling interpretation until an observation succeeds. With probability 1, the first successful test will be **observe**$(x \leq 0.3001 \;\&\&\; x \geq 0.3)$ and the program will therefore return false with probability 1.

A.5 Random Interval Covering Over $[0, 1]$

The discretized program for the case study from §2.6 is shown in Figure 20. Slice takes 0.000s, and we generate a Markov chain consisting of 192365 states in 38.69 seconds. Storm takes 0.69 seconds for model checking, and reports that the sought-after probability is 0.034474....

B Slice Case Studies (continued)

B.1 Random Interval Covering Over $[0, 1]$ with Conditioning

The full program for the case study from §7.1.2 is shown in Figure 21. The discretized program is shown in Figure 22.

```
(* Returns true if y is in [lo,hi) *)
let within = fun lo -> fun hi -> fun y ->
  (y >= lo) && (y < hi) in

(* Function for sampling interval bounds *)
let b = fun _ ->
  discrete(0.18: 0.05, 0.11: 0.24,
           0.21: 0.37, 0.09: 0.58,
           0.41: 0.89) in

(* Initialize intervals by sampling their
   respective (possibly overlapping) bounds *)
let init =
  (false, within 0.00 (b ())) ::
  (false, within (b ()) (b ())) ::
  (false, within (b ()) (b ())) ::
  (false, within (b ()) 1.00) ::
  nil in

(* Update flags depending on y falling
   within the respective interval *)
let update = fix update y := fun ys ->
  match ys with
    | nil -> nil
    | h :: t ->
        ((fst h || ((snd h) y)), snd h)
        :: (update y t)
  end in

(* Ensure sampled y hits at most one
   interval *)
let non_overlap =
  fix non_overlap y := fun seen -> fun ys ->
    match ys with
      | nil -> true
      | h :: t ->
          let hit = (snd h) y in
          if hit && seen then false
          else non_overlap y (seen || hit) t
    end in

(* Use the sampled value to update flags *)
let step = fun y -> fun ys ->
  update y ys in

(* Number of samples determined by
   unbounded recursion *)
let cover = fix cover st :=
  if uniform(0,1) < 0.1 then st
  else
    let y = gaussian(0,1) in
    let _ = observe(non_overlap y false st) in
    cover (step y st) in

(* Have all intervals been covered
   at least once? *)
let is_covered = fix is_covered ys :=
  match ys with
    | nil -> true
    | h :: t -> (fst h) && (is_covered t)
  end in

(* Run the process and query whether
   the whole interval was covered *)
let final = cover init in
  is_covered final
```

Fig. 21. An infinite stochastic process for random interval covering over $[0, 1]$, involving conditioning, continuous sampling, higher-order recursion, and lists.

B.2 Bloom Filter

The full program for the case study from §7.1.3 is shown in Figure 23. The discretized program is shown in Figure 24.

```
(* Returns true if y is in [lo,hi) *)
let within = fun lo -> fun hi -> fun y ->
  (y >= lo) && (y < hi) in

(* Function for sampling interval bounds *)
let b = fun _ ->
  discrete(0.18: 0.05, 0.11: 0.24,
           0.21: 0.37, 0.09: 0.58,
           0.41: 0.89) in

(* Initialize intervals by sampling their
   respective (possibly overlapping) bounds *)
let init =
  (false, within 1 (b ())) ::
  (false, within (b ()) (b ())) ::
  (false, within (b ()) (b ())) ::
  (false, within (b ()) 8) ::
  nil in

(* Update flags depending on y falling
   within the respective interval *)
let update = fix update y := fun ys ->
  match ys with
    | nil -> nil
    | h :: t ->
        ((fst h || ((snd h) y)), snd h)
        :: (update y t)
  end in

(* Ensure sampled y hits at most one
   interval *)
let non_overlap =
  fix non_overlap y := fun seen -> fun ys ->
    match ys with
      | nil -> true
      | h :: t ->
          let hit = (snd h) y in
          if hit && seen then false
          else non_overlap y (seen || hit) t
    end in

(* Use the sampled value to update flags *)
let step = fun y -> fun ys ->
  update y ys in

(* Number of samples determined by
   unbounded recursion *)
let cover = fix cover st :=
  if discrete(0.1, 0.9) < 1 then st
  else
    let y = discrete(0.5, 0.0199388,
                     0.0748961, 0.0494739,
                     0.0747339, 0.0632619,
                     0.0309625, 0.0280777,
                     0.158655) in
    let _ = observe(non_overlap y false st) in
    cover (step y st) in

(* Have all intervals been covered
   at least once? *)
let is_covered = fix is_covered ys :=
  match ys with
    | nil -> true
    | h :: t -> (fst h) && (is_covered t)
  end in

(* Run the process and query whether
   the whole interval was covered *)
let final = cover init in
  is_covered final
```

Fig. 22. A *discretized* infinite stochastic process for random interval covering over $[0, 1]$, involving conditioning, continuous sampling, higher-order recursion, and lists.

```
(* Set bf[idx] := true, where idx is 0..3 and
   bf = (b0, (b1, (b2, b3))) *)
let set_true = fun bf -> fun idx ->
  let b0 = fst bf in
  let tail1 = snd bf in
  let b1 = fst tail1 in
  let tail2 = snd tail1 in
  let b2 = fst tail2 in
  let b3 = snd tail2 in
  if idx <= 0 then
    (true, (b1, (b2, b3)))
  else if idx <= 1 then
    (b0, (true, (b2, b3)))
  else if idx <= 2 then
    (b0, (b1, (true, b3)))
  else
    (b0, (b1, (b2, true))) in

(* Get bf[idx], where idx is 0..3 *)
let get = fun bf -> fun idx ->
  let b0 = fst bf in
  let tail1 = snd bf in
  let b1 = fst tail1 in
  let tail2 = snd tail1 in
  let b2 = fst tail2 in
  let b3 = snd tail2 in
  if idx <= 0 then b0
  else if idx <= 1 then b1
  else if idx <= 2 then b2
  else b3 in

(* Insert element q into bf using hash
   functions h1 and h2 *)
let insert = fun q -> fun h1
            -> fun h2 -> fun bf ->
  let index1 = h1 q in
  let index2 = h2 q in
  let bf1 = set_true bf index1 in
  let bf2 = set_true bf1 index2 in
  bf2 in

(* Search for q in bf using hash
   functions h1 and h2 *)
let search = fun q -> fun h1
            -> fun h2 -> fun bf ->
  let index1 = h1 q in
  let index2 = h2 q in
  let b01 = get bf index1 in
  let b02 = get bf index2 in
  b01 && b02 in

(* Hash functions *)
let h1 = fun q ->
  if q <= -2 then 0
  else if -2 < q && q <= 0 then 1
  else if 0 < q && q <= 2 then 2
  else 3 in

let h2 = fun q ->
  if q <= -2 then 3
  else if -2 < q && q <= 0 then 2
  else if 0 < q && q <= 2 then 1
  else 0 in

(* Test whether the Bloom filter is full *)
let full = fun bf ->
  let b0 = fst bf in
  let t1 = snd bf in
  let b1 = fst t1 in
  let t2 = snd t1 in
  let b2 = fst t2 in
  let b3 = snd t2 in
  b0 && b1 && b2 && b3 in

(* Initial, empty Bloom filter *)
let init = (false, (false, (false, false))) in

(* Query key *)
let key = gaussian(0,1) in

(* Insert gaussian samples until bf is full
   or the key is reported present *)
let search_before_full = fix loop bf :=
  if full bf then bf
  else if search key h1 h2 bf then bf
  else
    let q = gaussian(0,1) in
    loop (insert q h1 h2 bf) in

(* Query whether search terminated before
   the filter became full *)
let bf_final = search_before_full init in
  not (full bf_final)
```

Fig. 23. A 4-bit Bloom filter program with continuous samples, locality-sensitive hash functions, and search and insert functions.

```
(* Set bf[idx] := true, where idx is 0..3 and
   bf = (b0, (b1, (b2, b3))) *)
let set_true = fun bf -> fun idx ->
  let b0 = fst bf in
  let tail1 = snd bf in
  let b1 = fst tail1 in
  let tail2 = snd tail1 in
  let b2 = fst tail2 in
  let b3 = snd tail2 in
  if idx <= 0 then
    (true, (b1, (b2, b3)))
  else if idx <= 1 then
    (b0, (true, (b2, b3)))
  else if idx <= 2 then
    (b0, (b1, (true, b3)))
  else
    (b0, (b1, (b2, true))) in

(* Get bf[idx], where idx is 0..3 *)
let get = fun bf -> fun idx ->
  let b0 = fst bf in
  let tail1 = snd bf in
  let b1 = fst tail1 in
  let tail2 = snd tail1 in
  let b2 = fst tail2 in
  let b3 = snd tail2 in
  if idx <= 0 then b0
  else if idx <= 1 then b1
  else if idx <= 2 then b2
  else b3 in

(* Insert element q into bf using hash
   functions h1 and h2 *)
let insert = fun q -> fun h1
            -> fun h2 -> fun bf ->
  let index1 = h1 q in
  let index2 = h2 q in
  let bf1 = set_true bf index1 in
  let bf2 = set_true bf1 index2 in
  bf2 in

(* Search for q in bf using hash
   functions h1 and h2 *)
let search = fun q -> fun h1
            -> fun h2 -> fun bf ->
  let index1 = h1 q in
  let index2 = h2 q in
  let b01 = get bf index1 in
  let b02 = get bf index2 in
  b01 && b02 in

(* Hash functions *)
let h1 = fun q ->
  if q <= 0 then 0
  else if 0 < q && q <= 1 then 1
  else if 1 < q && q <= 2 then 2
  else 3 in

let h2 = fun q ->
  if q <= 0 then 3
  else if 0 < q && q <= 1 then 2
  else if 1 < q && q <= 2 then 1
  else 0 in

(* Test whether the Bloom filter is full *)
let full = fun bf ->
  let b0 = fst bf in
  let t1 = snd bf in
  let b1 = fst t1 in
  let t2 = snd t1 in
  let b2 = fst t2 in
  let b3 = snd t2 in
  b0 && b1 && b2 && b3 in

(* Initial, empty Bloom filter *)
let init = (false, (false, (false, false))) in

(* Query key *)
let key = discrete(0.0227501,
                   0.47725,
                   0.47725,
                   0.0227501) in

(* Insert discretized gaussian samples until
   bf is full or the key is reported
   present *)
let search_before_full = fix loop bf :=
  if full bf then bf
  else if search key h1 h2 bf then bf
  else
    let q = discrete(0.0227501, 0.47725,
                     0.47725, 0.0227501) in
    loop (insert q h1 h2 bf) in

(* Query whether search terminated before
   the filter became full *)
let bf_final = search_before_full init in
  not (full bf_final)
```

Fig. 24. A *discretized* 4-bit Bloom filter program with continuous samples, locality-sensitive hash functions, and search and insert functions.

Source constructor	Density or PMF	Support	Parameter domain and convention
Continuous distributions			
`exponential(`λ`)`	$\lambda e^{-\lambda x}$	$[0, \infty)$	Rate $\lambda > 0$
`uniform(`a, b`)`	$\frac{1}{b-a}\mathbf{1}_{[a,b)}(x)$	$[a, b]$	Endpoints $a < b$
`laplace(`b`)`	$\frac{1}{2b}e^{-\lvert x\rvert/b}$	$\mathbb{R}$	Centered; scale $b > 0$
`gaussian(`μ, σ`)`	$\frac{1}{\sqrt{2\pi}\sigma}e^{-(x-\mu)^2/(2\sigma^2)}$	$\mathbb{R}$	Location $\mu \in \mathbb{R}$; standard deviation $\sigma > 0$
`cauchy(`γ`)`	$\frac{\gamma}{\pi(x^2+\gamma^2)}$	$\mathbb{R}$	Centered; scale $\gamma > 0$
`beta(`α, β`)`	$\frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)}$	$[0, 1]$	Shapes $\alpha, \beta > 0$
`tdist(`ν`)`	$\frac{\Gamma((\nu+1)/2)}{\sqrt{\nu\pi}\,\Gamma(\nu/2)}(1 + x^2/\nu)^{-(\nu+1)/2}$	$\mathbb{R}$	Centered; degrees of freedom $\nu > 0$
`lognormal(`μ, σ`)`	$\frac{1}{x\sigma\sqrt{2\pi}}e^{-(\log x-\mu)^2/(2\sigma^2)}$	$[0, \infty)$	Log-location $\mu \in \mathbb{R}$; log-scale $\sigma > 0$
`chi2(`ν`)`	$\frac{x^{\nu/2-1}e^{-x/2}}{2^{\nu/2}\Gamma(\nu/2)}$	$[0, \infty)$	Real degrees of freedom $\nu > 0$
`gamma(`α, θ`)`	$\frac{x^{\alpha-1}e^{-x/\theta}}{\Gamma(\alpha)\theta^{\alpha}}$	$[0, \infty)$	Shape $\alpha > 0$; scale $\theta > 0$
`logistic(`s`)`	$\frac{e^{-x/s}}{s(1+e^{-x/s})^2}$	$\mathbb{R}$	Centered; scale $s > 0$
`pareto(`a, b`)`	$ab^a x^{-a-1}$	$[b, \infty)$	GSL shape $a > 0$; threshold/scale $b > 0$
`rayleigh(`σ`)`	$\frac{x}{\sigma^2}e^{-x^2/(2\sigma^2)}$	$[0, \infty)$	Scale $\sigma > 0$
`weibull(`a, b`)`	$\frac{b}{a}(x/a)^{b-1}e^{-(x/a)^b}$	$[0, \infty)$	GSL scale $a > 0$; exponent $b > 0$
`gumbel1(`a, b`)`	$ab\,e^{-(be^{-ax}+ax)}$	$\mathbb{R}$	GSL parameters $a, b > 0$
`gumbel2(`a, b`)`	$ab\,x^{-a-1}e^{-bx^{-a}}$	$[0, \infty)$	GSL shape $a > 0$; coefficient $b > 0$
`exppow(`a, b`)`	$\frac{1}{2a\Gamma(1+1/b)}e^{-(\lvert x\rvert/a)^b}$	$\mathbb{R}$	Scale $a > 0$; exponent $b > 0$
Discrete distributions			
`poisson(`μ`)`	$e^{-\mu}\mu^k/k!$	$\mathbb{N}_0$ if $\mu > 0$; $\{0\}$ if $\mu = 0$	Mean $\mu \geq 0$
`binomial(`p, n`)`	$\binom{n}{k}p^k(1-p)^{n-k}$	$\{0, \ldots, n\}$, with endpoint degeneracies	$0 \leq p \leq 1$; integer $0 \leq n \leq \min(2^{32}-1, \texttt{max\_int})$

Fig. 25. List of supported distributions. The binomial support is $\{0\}$ when $n = 0$ or $p = 0$, and $\{n\}$ when $p = 1$; otherwise it is $\{0, \ldots, n\}$.

C List of Supported Slice Distributions

The list of distributions supported by Slice, along with their probability density / mass functions, is given in Figure 25. Slice supports discrete distributions by applying the same cut-set and value-set analysis used for continuous distributions, *provided that the discrete distribution admits a well-defined CDF*. The extension to discrete distributions does not introduce any fundamental changes to the discretization algorithm.

D Typing Rules for Full Slice Language

The typing rules for general language constructs is given in Figure 26.

$$\textsc{Var}\ \frac{}{\Gamma, x : \tau \vdash x : \tau} \qquad \textsc{Let}\ \frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \mathbf{let}\ x = e_1\ \mathbf{in}\ e_2 : \tau_2} \qquad \textsc{If}\ \frac{\Gamma \vdash e_1 : \mathbf{bool} \qquad \Gamma \vdash e_2 : \tau \qquad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \mathbf{if}\ e_1\ \mathbf{then}\ e_2\ \mathbf{else}\ e_3 : \tau}$$

$$\textsc{Discrete}\ \frac{n \geq 1 \qquad \forall i \in \{0, \ldots, n-1\}.\ p_i \geq 0 \qquad \sum_{i=0}^{n-1} p_i = 1}{\Gamma \vdash \mathbf{discrete}(p_0, \ldots, p_{n-1}) : \mathbf{fin}(n)} \qquad \textsc{LessEq}\ \frac{\Gamma \vdash e : \mathbf{int}}{\Gamma \vdash e \leq i : \mathbf{bool}}$$

$$\textsc{Pair}\ \frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 * \tau_2} \qquad \textsc{Fst}\ \frac{\Gamma \vdash e : \tau_1 * \tau_2}{\Gamma \vdash \mathbf{fst}\ e : \tau_1} \qquad \textsc{Snd}\ \frac{\Gamma \vdash e : \tau_1 * \tau_2}{\Gamma \vdash \mathbf{snd}\ e : \tau_2} \qquad \textsc{Fun}\ \frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \mathbf{fun}\ x\ \rightarrow\ e : \tau_1 \rightarrow \tau_2}$$

$$\textsc{App}\ \frac{\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \qquad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2} \qquad \textsc{True}\ \frac{}{\Gamma \vdash \mathrm{true} : \mathbf{bool}} \qquad \textsc{False}\ \frac{}{\Gamma \vdash \mathrm{false} : \mathbf{bool}}$$

$$\textsc{And}\ \frac{\Gamma \vdash e_1 : \mathbf{bool} \qquad \Gamma \vdash e_2 : \mathbf{bool}}{\Gamma \vdash e_1\ \&\&\ e_2 : \mathbf{bool}} \qquad \textsc{Or}\ \frac{\Gamma \vdash e_1 : \mathbf{bool} \qquad \Gamma \vdash e_2 : \mathbf{bool}}{\Gamma \vdash e_1\ ||\ e_2 : \mathbf{bool}} \qquad \textsc{Not}\ \frac{\Gamma \vdash e : \mathbf{bool}}{\Gamma \vdash \mathrm{not}\ e : \mathbf{bool}}$$

$$\textsc{Unit}\ \frac{}{\Gamma \vdash () : \mathbf{unit}} \qquad \textsc{FinConst}\ \frac{0 \leq k < n}{\Gamma \vdash k_{\#n} : \mathbf{fin}(n)} \qquad \textsc{FinLess}\ \frac{\Gamma \vdash e_1 : \mathbf{fin}(n) \qquad \Gamma \vdash e_2 : \mathbf{fin}(n)}{\Gamma \vdash e_1 <_{\#n} e_2 : \mathbf{bool}}$$

$$\textsc{FinLessEq}\ \frac{\Gamma \vdash e_1 : \mathbf{fin}(n) \qquad \Gamma \vdash e_2 : \mathbf{fin}(n)}{\Gamma \vdash e_1 \leq_{\#n} e_2 : \mathbf{bool}} \qquad \textsc{Observe}\ \frac{\Gamma \vdash e : \mathbf{bool}}{\Gamma \vdash \mathbf{observe}\ e : \mathbf{unit}} \qquad \textsc{Seq}\ \frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1; e_2 : \tau_2}$$

$$\textsc{Fix}\ \frac{\Gamma, f : \tau_1 \rightarrow \tau_2, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \mathrm{fix}\ f\ x := e : \tau_1 \rightarrow \tau_2} \qquad \textsc{Nil}\ \frac{}{\Gamma \vdash \mathrm{nil} : \mathbf{list}(\tau)} \qquad \textsc{Cons}\ \frac{\Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \mathbf{list}(\tau)}{\Gamma \vdash e_1 :: e_2 : \mathbf{list}(\tau)}$$

$$\textsc{Match}\ \frac{\Gamma \vdash e : \mathbf{list}(\tau_1) \qquad \Gamma \vdash e_1 : \tau_2 \qquad \Gamma, h : \tau_1, t : \mathbf{list}(\tau_1) \vdash e_2 : \tau_2}{\Gamma \vdash \mathrm{match}\ e\ \mathrm{with\ nil} \rightarrow e_1 \mid h :: t \rightarrow e_2\ \mathrm{end} : \tau_2} \qquad \textsc{Diverge}\ \frac{}{\Gamma \vdash \mathbf{diverge} : \tau}$$

Fig. 26. Typing rules for general language constructs

E Discretization Proof

E.1 Well-Typedness of Discretization

Proof of Theorem 5.4. We proceed by induction on the structure of the typing derivation $\Gamma \vdash e : \tau$, matching each case of the discretization function $\mathcal{D}[\![\cdot]\!]$ defined in Figure 13.

Case $e = x$.

Since $\Gamma(x) = \tau$, we have $\mathcal{D}[\![\Gamma]\!](x) = \mathcal{D}[\![\tau]\!]$. Moreover, $\mathcal{D}[\![x : \tau]\!] = x$, so $\mathcal{D}[\![\Gamma]\!] \vdash x : \mathcal{D}[\![\tau]\!]$.

Case $e = (c : \mathbf{float}[B; V])$.

The Float-Const rule was applied, so $\mathrm{has}(V, c)$ holds.

- If $B = \top$: then $\mathcal{D}[\![c : \mathbf{float}[B; V]]\!] = c$ and $\mathcal{D}[\![\mathbf{float}[B; V]]\!] = \mathbf{float}[B; V]$. Since $\mathrm{has}(V, c)$ still holds, c is well-typed at $\mathbf{float}[B; V]$ by Float-Const.
- If $B \neq \top$: then $\mathcal{D}[\![c : \mathbf{float}[B; V]]\!] = k_{\#|B|+1}$ for the unique k such that $c \in \mathrm{Intervals}(B)_k$, and $\mathcal{D}[\![\mathbf{float}[B; V]]\!] = \mathbf{fin}(|B| + 1)$. Since $0 \leq k < |B| + 1$, the finite constant $k_{\#|B|+1}$ is well-typed at $\mathbf{fin}(|B| + 1)$ by FinConst.

Case $e = (\mathbf{let}\ x = e_1 : \tau_1\ \mathbf{in}\ e_2 : \tau_2)$.

By the inductive hypothesis,

$$\mathcal{D}[\![\Gamma]\!] \vdash \mathcal{D}[\![e_1 : \tau_1]\!] : \mathcal{D}[\![\tau_1]\!] \quad \text{and} \quad \mathcal{D}[\![\Gamma, x : \tau_1]\!] \vdash \mathcal{D}[\![e_2 : \tau_2]\!] : \mathcal{D}[\![\tau_2]\!].$$

Note that $\mathcal{D}[\![\Gamma, x : \tau_1]\!] = \mathcal{D}[\![\Gamma]\!], x : \mathcal{D}[\![\tau_1]\!]$. Applying Let gives

$$\mathcal{D}[\![\Gamma]\!] \vdash \mathbf{let}\ x = \mathcal{D}[\![e_1 : \tau_1]\!]\ \mathbf{in}\ \mathcal{D}[\![e_2 : \tau_2]\!] : \mathcal{D}[\![\tau_2]\!].$$

Case $e = (\mathbf{if}\ e_1 : \mathbf{bool}\ \mathbf{then}\ e_2 : \tau\ \mathbf{else}\ e_3 : \tau)$.

By the inductive hypothesis, $\mathcal{D}[\![\Gamma]\!] \vdash \mathcal{D}[\![e_i : \tau_i]\!] : \mathcal{D}[\![\tau_i]\!]$ for $i = 1, 2, 3$, where $\tau_1 = \mathbf{bool}$ and $\tau_2 = \tau_3 = \tau$. Since $\mathcal{D}[\![\mathbf{bool}]\!] = \mathbf{bool}$, applying If gives

$$\mathcal{D}[\![\Gamma]\!] \vdash \mathbf{if}\ \mathcal{D}[\![e_1 : \mathbf{bool}]\!]\ \mathbf{then}\ \mathcal{D}[\![e_2 : \tau]\!]\ \mathbf{else}\ \mathcal{D}[\![e_3 : \tau]\!] : \mathcal{D}[\![\tau]\!].$$

Case $e = (e_1 : \mathbf{float}[B; V_1] < e_2 : \mathbf{float}[B; V_2])$.

The Less rule was applied, so both subexpressions share the same cut set B and $\mathrm{answers}_<(B, V_1, V_2)$ holds. By the inductive hypothesis, $\mathcal{D}[\![\Gamma]\!] \vdash \mathcal{D}[\![e_i : \mathbf{float}[B; V_i]]\!] : \mathcal{D}[\![\mathbf{float}[B; V_i]]\!]$ for $i = 1, 2$.

- If $B = \top$: then $\mathcal{D}[\![\mathbf{float}[B; V_i]]\!] = \mathbf{float}[B; V_i]$ for both i, and the standard float comparison gives type $\mathbf{bool} = \mathcal{D}[\![\mathbf{bool}]\!]$.
- If $B \neq \top$: then both subexpressions are discretized to type $\mathbf{fin}(|B|+1)$, and the finite comparison $<_{\#(|B|+1)}$ is well-typed at $\mathbf{bool} = \mathcal{D}[\![\mathbf{bool}]\!]$ by FinLess.

Case $e = (e_1 : \mathbf{float}[B; V_1] \leq e_2 : \mathbf{float}[B; V_2])$.

Analogous to the $<$ case, using $\mathrm{answers}_\leq(B, V_1, V_2)$ and FinLessEq.

Case $e = (\mathbf{uniform}(e_1 : \mathbf{float}[B_1; V_1],\ e_2 : \mathbf{float}[B_2; V_2]) : \mathbf{float}[B; \top])$.

The Continuous-2 rule was applied, so $\mathrm{recovers}(B_1, V_1)$ and $\mathrm{recovers}(B_2, V_2)$ hold. By the inductive hypothesis, $\mathcal{D}[\![\Gamma]\!] \vdash \mathcal{D}[\![e_i : \mathbf{float}[B_i; V_i]]\!] : \mathcal{D}[\![\mathbf{float}[B_i; V_i]]\!]$ for $i = 1, 2$. Also, $B_i \equiv_\top B$, so $B_i = \top$ iff $B = \top$. We consider each sub-case of the discretization function:

- $e_1, e_2 \in \mathbb{R}$ and $e_1 \geq e_2$: then $\mathcal{D}[\![\texttt{uniform}(\ldots) : \texttt{float}[B;\top]]\!] = \texttt{diverge}$, which is well-typed at any type by Diverge.
- $B = \top$: then $\mathcal{D}[\![\texttt{float}[B;\top]]\!] = \texttt{float}[B;\top]$, and

$\mathcal{D}[\![\texttt{uniform}(e_1, e_2) : \texttt{float}[B;\top]]\!] = \texttt{uniform}(\mathcal{D}[\![e_1 : \texttt{float}[B_1;V_1]]\!], \mathcal{D}[\![e_2 : \texttt{float}[B_2;V_2]]\!])$.

  Here $B_1 = B_2 = \top$, so the inductive hypotheses give float-typed arguments; Continuous-2 yields $\texttt{float}[\top;\top]$.

- $B \neq \top$ and $e_1, e_2 \in \mathbb{R}$: then $\mathcal{D}[\![\texttt{float}[B;\top]]\!] = \texttt{fin}(|B|+1)$, and

$$\mathcal{D}[\![\texttt{uniform}(e_1, e_2) : \texttt{float}[B;\top]]\!] = \texttt{discrete}(p_0, \ldots, p_{|B|})$$

  where $p_k = \mathrm{CDF}(b_{k+1}) - \mathrm{CDF}(b_k)$. Each $p_k \in [0,1]$ and $\sum_k p_k = 1$, so this is well-typed at $\texttt{fin}(|B|+1)$ by Discrete.

- If $B, B_1, B_2 \neq \top$ and $V_1 = \emptyset$ or $V_2 = \emptyset$, then the enumeration over $V_1 \times V_2$ is empty, so the generated expression evaluates $\mathcal{D}[\![e_1]\!]$ and $\mathcal{D}[\![e_2]\!]$ through its two outer **let**-bindings and then yields **diverge**; the bindings are well-typed by the inductive hypotheses and weakening, and their body is well-typed at $\texttt{fin}(|B|+1)$ by Diverge.
- $B, B_1, B_2 \neq \top$, $V_1, V_2 \neq \emptyset$, and $e_1 \notin \mathbb{R}$ or $e_2 \notin \mathbb{R}$: the algorithm generates conditionals over all pairs (v_i, w_j), each branch dispatching to a constant parameter in **uniform**. The inductive hypotheses and weakening type the two outer **let**-bindings. In each guard, both operands of each finite equality have the same finite type; since finite equality is desugared using FinLessEq, each equality has type **bool**, as does their conjunction. By a previous subcase, each branch $\mathcal{D}[\![\texttt{uniform}(v_i, w_j) : \texttt{float}[B;\top]]\!]$ is well-typed at $\texttt{fin}(|B|+1) = \mathcal{D}[\![\texttt{float}[B;\top]]\!]$. The final **else diverge** branch has the same type by Diverge; hence the conditional structure is well-typed by If (as in the **if** case above), and the full expression is well-typed at $\mathcal{D}[\![\texttt{float}[B;\top]]\!]$.

Case a one-argument continuous distribution. Analogous, using $B_1 \equiv_\top B$.

Case e is any other language construct (functions, pairs, lists, *etc.*).

By definition of $\mathcal{D}[\![\cdot]\!]$, the discretization recurses structurally over e and τ. That is, $\mathcal{D}[\![e : \tau]\!]$ is formed by applying $\mathcal{D}[\![\cdot]\!]$ to each subexpression and subtype of $e : \tau$, and $\mathcal{D}[\![\tau]\!]$ is formed by applying $\mathcal{D}[\![\cdot]\!]$ to each component type of τ. The result then follows immediately by applying the inductive hypothesis to each subterm, analogous to the above cases.

In every case, the typing derivation is preserved under $\mathcal{D}[\![\cdot]\!]$. Therefore,

$$\mathcal{D}[\![\Gamma]\!] \vdash \mathcal{D}[\![e : \tau]\!] : \mathcal{D}[\![\tau]\!]. \qquad \square$$

F Soundness Proof

F.1 Measure Theory Preliminaries

We first give a brief overview of basic measure-theoretic terms and lemmas, and refer the reader to standard textbooks for further background [43, 44].

Topological space. A topological space is a pair $(X, \mathcal{T})$ where $\mathcal{T}$ is a collection of subsets of X, called open sets, such that $\emptyset, X \in \mathcal{T}$, and $\mathcal{T}$ is closed under arbitrary unions and finite intersections.

σ-algebra. A σ-algebra on a set X is a collection Σ of subsets of X that includes X itself and is closed under complement and under countable unions. By implication, a σ-algebra includes the empty set $\emptyset$ and is closed under countable intersections. Given a topological space $(X, \mathcal{T})$, the Borel σ-algebra $\mathcal{B}(X)$ is the smallest σ-algebra containing all open sets in $\mathcal{T}$.

Measurable space. A measurable space is a pair (X, Σ) consisting of a set X and a σ-algebra Σ on X. When Σ is the Borel σ-algebra on a topological space X, we call (X, Σ) a Borel measurable space.

Measure. Let (X, Σ) be a measurable space. A function $\mu : \Sigma \to [0, \infty]$ is called a measure if it satisfies the following properties: nonnegativity, $\mu(\emptyset) = 0$, and countable additivity. When Σ is a Borel σ-algebra, μ is called a Borel measure.

Measure space. The triple (X, Σ, μ) is called a measure space. When Σ is the Borel σ-algebra and μ is a Borel measure, (X, Σ, μ) is called a Borel measure space.

Subprobability measure. Let (X, Σ, μ) be a measure space. When $\mu(X) \leq 1$, μ is called a (Borel) subprobability measure. When $\mu(X) = 1$, μ is called a (Borel) probability measure and (X, Σ, μ) a (Borel) probability space.

Product measurable space. The product of two measurable spaces (X_1, Σ_1) and (X_2, Σ_2) is also a measurable space, namely $(X_1 \times X_2, \Sigma_1 \times \Sigma_2)$, where $\Sigma_1 \times \Sigma_2$ is the σ-algebra generated by sets of the form $A_1 \times A_2$ with $A_1 \in \Sigma_1$ and $A_2 \in \Sigma_2$.

Generated σ-algebra. Let X be a set and let $\mathcal{G} \subseteq \mathcal{P}(X)$ be a collection of subsets of X. The σ-algebra generated by $\mathcal{G}$, denoted $\sigma(\mathcal{G})$, is the smallest σ-algebra on X that contains $\mathcal{G}$. Equivalently, it is the intersection of all σ-algebras on X that contain $\mathcal{G}$.

Subprobability distribution. Let (X, Σ) be a measurable space. We write $\mathcal{D}(X)$ for the set of all subprobability measures on (X, Σ).

Dirac measure. Let (X, Σ) be a measurable space and let $x \in X$. The *Dirac measure* at x is the probability measure

$$\delta_x : \Sigma \to [0, 1]$$

defined by

$$\delta_x(A) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A. \end{cases}$$

Measurable function. A function $f : X \to Y$ between measurable spaces $(X, \mathcal{F}_X)$ and $(Y, \mathcal{F}_Y)$ is called measurable if the preimage of any measurable set in Y is measurable in X, i.e. if

$$f^{-1}(A) = \{x \in X \mid f(x) \in A\} \in \mathcal{F}_X$$

for all $A \in \mathcal{F}_Y$.

Definition F.1 (Subspace σ-algebra). Let (X, Σ) be a Borel measurable space and let $Y \subseteq X$. The subspace (or trace) σ-algebra on Y is defined as

$$\Sigma_Y = \{Y \cap A \mid A \in \Sigma\}.$$

Then (Y, Σ_Y) is a Borel measurable space.

Definition F.2 (σ-algebra generated by a function). Let $f : X \to Y$ be a function and let (Y, Σ_Y) be a measurable space. The σ-algebra generated by f is defined by:

$$f^{-1}(\Sigma_Y) := \{f^{-1}(A) \mid A \in \Sigma_Y\}.$$

Then $(X, f^{-1}(\Sigma_Y))$ is a measurable space.

Lemma F.3 (Bijection induces isomorphism). *Let $f : X \to Y$ be a bijection and let (Y, Σ_Y) be a measurable space. Equip X with the σ-algebra generated by f:*

$$\Sigma_X \;:=\; f^{-1}(\Sigma_Y) \;=\; \{f^{-1}(A) \mid A \in \Sigma_Y\}.$$

Then,

(1) (X, Σ_X) is a measurable space, and
(2) f induces an isomorphism of measurable spaces:

$$(X, \Sigma_X) \cong (Y, \Sigma_Y).$$

Proof. We first verify that i) (X, Σ_X) is a measurable space, then show that ii) f induces an isomorphism of measurable spaces $(X, \Sigma_X) \cong (Y, \Sigma_Y)$.

Proof of i) We show that $\Sigma_X = f^{-1}(\Sigma_Y)$ is a σ-algebra on X.

- **(Contains X):** $X = f^{-1}(Y) \in \Sigma_X$ since $Y \in \Sigma_Y$.
- **(Complements):** If $f^{-1}(A) \in \Sigma_X$, then $X \setminus f^{-1}(A) = f^{-1}(Y \setminus A) \in \Sigma_X$ since $Y \setminus A \in \Sigma_Y$.
- **(Countable unions):** If $f^{-1}(A_i) \in \Sigma_X$ for $i \in \mathbb{N}$, then $\bigcup_i f^{-1}(A_i) = f^{-1}(\bigcup_i A_i) \in \Sigma_X$ since $\bigcup_i A_i \in \Sigma_Y$.

Hence (X, Σ_X) is a measurable space.

Proof of ii) Two measurable spaces (X, Σ_X) and (Y, Σ_Y) are isomorphic iff there exists a bijection $f : X \to Y$ such that f and f^{-1} are measurable.

We first show that f is measurable. Measurability of f means that $f^{-1}(A) \in \Sigma_X$ for every $A \in \Sigma_Y$. Let $A \in \Sigma_Y$ be arbitrary. By definition of Σ_X,

$$f^{-1}(A) \in f^{-1}(\Sigma_Y) = \Sigma_X.$$

It follows that f is measurable.

Next we show that f^{-1} is measurable. Measurability of f^{-1} means that $(f^{-1})^{-1}(B) \in \Sigma_Y$ for every $B \in \Sigma_X$. Let $B \in \Sigma_X$ be arbitrary. By definition of Σ_X, there exists some $A \in \Sigma_Y$ such that $B = f^{-1}(A)$.

Observe that:

$$\begin{aligned}
(f^{-1})^{-1}(B) &= f(B) && \text{(preimage of } f^{-1} \text{ is } f\text{)} \\
&= f(f^{-1}(A)) && \text{(substituting } B = f^{-1}(A)\text{)} \\
&= A && (f \text{ is a bijection})
\end{aligned}$$

Since $A \in \Sigma_Y$, it follows that f^{-1} is measurable.

Therefore f is a measurable bijection with measurable inverse, so it is an isomorphism of measurable spaces. □

Lemma F.4 (Minimality of generated σ-algebras). *Let X be a set and let $\mathcal{G} \subseteq \mathcal{P}(X)$. Suppose C is a σ-algebra on X such that $\mathcal{G} \subseteq C$. Then*

$$\sigma(\mathcal{G}) \subseteq C.$$

Proof. Recall that $\sigma(\mathcal{G})$ is defined as the smallest σ-algebra on X containing $\mathcal{G}$, given concretely by the intersection of all σ-algebras on X containing $\mathcal{G}$:

$$\sigma(\mathcal{G}) = \bigcap_{\substack{\mathcal{F} \text{ is a } \sigma\text{-algebra on } X \\ \mathcal{G} \subseteq \mathcal{F}}} \mathcal{F}.$$

Since C is a σ-algebra on X containing $\mathcal{G}$, it appears in this intersection. Therefore,

$$\sigma(\mathcal{G}) \subseteq C.$$

□

Definition F.5 (Sub-Markov kernels). Given measurable spaces (X, Σ_X) and (Y, Σ_Y), a *sub-Markov kernel* from X to Y is a function

$$P : X \times \Sigma_Y \to [0, 1]$$

such that:

(1) for every $x \in X$, $P(x, -)$ is a subprobability measure on Y;
(2) for every $A \in \Sigma_Y$, the map $x \mapsto P(x, A)$ is measurable.

A Markov kernel is a sub-Markov kernel satisfying $P(x, Y) = 1$ for every $x \in X$.

Definition F.6 (Projection with error). Let $(X + \bot, \Sigma_{X+\bot})$ and $(Y + \bot, \Sigma_{Y+\bot})$ be measurable spaces, with (X, Σ_X) and (Y, Σ_Y) measurable spaces as well. Let $\mu \in \mathcal{D}((X \times Y) + \bot)$. We define the first and second projection measures as:

(1) $\pi_1(\mu)(A) := \mu(A \times Y)$ for every measurable $A \in \Sigma_X$, and $\pi_1(\mu)(\bot) := \mu(\bot)$
(2) $\pi_2(\mu)(B) := \mu(X \times B)$ for every measurable $B \in \Sigma_Y$, and $\pi_2(\mu)(\bot) := \mu(\bot)$

Then, $\pi_1(\mu)$ is a measure on $(X + \bot, \Sigma_{X+\bot})$ and $\pi_2(\mu)$ is a measure on $(Y + \bot, \Sigma_{Y+\bot})$.

F.2 Measure Space on Expressions

To make the probabilistic semantics precise, we construct an appropriate measure space structure on the set of expressions by factoring out real numbers using the notion of what we call *expression skeletons*.

Definition F.7 (Expression Skeleton). An expression skeleton s is a syntactic expression obtained from a program by replacing each occurrence of a float constant with a hole of the form □. The set of skeletons is generated by the grammar:

$$\begin{aligned}
s ::= {} & \square \mid x \mid \text{true} \mid \text{false} \mid k_{\#n} \mid \textbf{diverge} \mid () \\
& \mid \textbf{let}\ x = s_1\ \textbf{in}\ s_2 \mid \textbf{if}\ s_1\ \textbf{then}\ s_2\ \textbf{else}\ s_3 \\
& \mid s_1 < s_2 \mid s_1 \le s_2 \mid s_1 <_{\#n} s_2 \mid s_1 \le_{\#n} s_2 \\
& \mid \textbf{uniform}(s_1, s_2) \mid \textbf{gaussian}(s_1, s_2) \mid \textbf{exponential}(s) \mid \textbf{beta}(s_1, s_2) \mid \textbf{discrete}(p_0, \ldots, p_{n-1}) \\
& \mid s_1\ \&\&\ s_2 \mid s_1\ ||\ s_2 \mid \text{not } s \mid \textbf{observe}(s) \mid s_1; s_2 \\
& \mid (s_1, s_2) \mid \textbf{fst}\ s \mid \textbf{snd}\ s \\
& \mid \textbf{fun}\ x \to s \mid s_1\ s_2 \mid \textbf{fix}\ f\ x := s \\
& \mid \texttt{nil} \mid s_1 :: s_2 \\
& \mid (\textbf{match}\ s\ \textbf{with} \mid \texttt{nil} \to s_{\text{nil}} \mid h :: t \to s_{\text{cons}}\ \textbf{end})
\end{aligned}$$

We denote by *Skel* the set of all expression skeletons, and by $holes(s)$ the number of distinct holes in skeleton s.

Example F.8. The expression $\textbf{uniform}(0.0, 1.0) < 0.5$ has the skeleton $\textbf{uniform}(\square, \square) < \square$ with three holes.

Definition F.9 (Filling a skeleton). Let $s \in Skel$ and let $\vec{r} = (r_1, \ldots, r_{holes(s)}) \in \mathbb{R}^{holes(s)}$. Define the expression $s[\vec{r}] \in Expr$ to be the result of replacing each hole $\square_i$ in s with the corresponding real constant r_i, for $1 \le i \le holes(s)$.

To recover a *typed* notion of skeletons, we extend the typing judgment to skeletons by adding a distinguished typing rule for holes.

Definition F.10 (Well-typed expression skeleton). Define a typing judgment for skeletons, written $\Gamma \vdash s : \tau$, with the understanding that all typing rules from expressions are preserved for skeletons, and that holes are typed as $\mathbf{float}[B; V]$ according to the rule below:

$$\frac{}{\square : \mathbf{float}[B; V]}$$

We denote by $Skel_\tau$ the set of all well-typed expression skeletons, where:

$$Skel_\tau := \{s \in Skel \mid s : \tau\}.$$

Remark F.11. Both $Skel_\tau$ and $Expr_\tau$ are to be understood as the set of all well-typed skeletons and expressions, respectively, *equipped with their full subtype information.*

For example,

$$\square : \mathbf{float}[B; \{0.5\}] < \square : \mathbf{float}[B; \{0.5\}] \quad \text{and} \quad \square : \mathbf{float}[B; \{0.2\}] < \square : \mathbf{float}[B; \{0.8\}]$$

are distinct elements of $Skel_{\mathbf{bool}}$, while

$$0.5 : \mathbf{float}[B; \{0.5\}] < 0.5 : \mathbf{float}[B; \{0.5\}] \quad \text{and} \quad 0.2 : \mathbf{float}[B; \{0.2\}] < 0.8 : \mathbf{float}[B; \{0.8\}]$$

are distinct elements of $Expr_{\mathbf{bool}}$.

In particular, both $Skel_\tau$ and $Expr_\tau$ are in general uncountable, since float subtype annotations of the form $\mathbf{float}[B; V]$ may vary over uncountably many real values.

Definition F.12 (Decomposing an expression). The decomposition function

$$\mathrm{dec} \colon Expr_\tau \longrightarrow \bigcup_{s \in Skel_\tau} \left(\{s\} \times \mathbb{R}^{holes(s)} \right)$$

is defined as the inverse of filling a skeleton, i.e., for every expression $e : \tau$,

$$\mathrm{dec}(e) = (s, \vec{r}) \quad \text{such that} \quad e = s[\vec{r}].$$

The explicit recursive definition of dec is shown in Figures 27 and 28.

Definition F.13 (Hole assignments of a skeleton). For a given $s : \tau \in Skel_\tau$, define the set of hole assignments of s as:

$$A_{s:\tau} := \bigcup_{e:\tau \in Expr_\tau} \{\mathrm{dec}(e : \tau)_2 \mid \mathrm{dec}(e : \tau)_1 = s : \tau\} \subseteq \mathbb{R}^{holes(s)}$$

Intuitively, $A_{s;\tau}$ is the set of all real tuples that can be substituted into the holes of s to yield a well-typed expression of type τ.

Example F.14. Consider the skeleton

$$s : \tau = (\square : \mathbf{float}[\{<3.5, <4.5\}; \top] \ < \ \square : \mathbf{float}[\{<3.5, <4.5\}; \{3.5, 4.5\}]) : \mathbf{bool}.$$

This skeleton has two holes:

- The left hole has type $\mathbf{float}[\{<3.5, <4.5\}; \top]$, so it may be instantiated by any real number.
- The right hole has type $\mathbf{float}[\{<3.5, <4.5\}; \{3.5, 4.5\}]$, so it may only be instantiated by one of the values 3.5 or 4.5.

An expression $e : \mathbf{bool}$ has skeleton s iff it is of the form

$$e = c_1 < c_2$$

where $c_1 \in \mathbb{R}$ and $c_2 \in \{3.5, 4.5\}$.

$$
\mathrm{dec}(e:\tau) = \begin{cases}
(x:\tau, ()) & \text{if } e = x:\tau \\
(\mathrm{true}:\mathbf{bool}, ()) & \text{if } e = \mathrm{true}:\mathbf{bool} \\
(\mathrm{false}:\mathbf{bool}, ()) & \text{if } e = \mathrm{false}:\mathbf{bool} \\
(k_{\#n}:\mathbf{fin}(n), ()) & \text{if } e = k_{\#n}:\mathbf{fin}(n) \\
(\mathbf{diverge}:\tau, ()) & \text{if } e = \mathbf{diverge}:\tau \\
((): \mathbf{unit}, ()) & \text{if } e = ():\mathbf{unit} \\
((\square, \mathbf{float}[B;V]):\mathbf{float}[B;V], c) & \text{if } e = c:\mathbf{float}[B;V] \\
(\mathbf{discrete}(p_0,\ldots,p_{n-1}):\mathbf{fin}(n), ()) & \text{if } e = \mathbf{discrete}(p_0,\ldots,p_{n-1}):\mathbf{fin}(n) \\
(\mathbf{uniform}(\mathrm{dec}(e_1)_1, \mathrm{dec}(e_2)_1):\mathbf{float}[B;V], & \\
\quad (\mathrm{dec}(e_1)_2, \mathrm{dec}(e_2)_2)) & \text{if } e = \mathbf{uniform}(e_1,e_2):\mathbf{float}[B;V] \\
(\mathbf{gaussian}(\mathrm{dec}(e_1)_1, \mathrm{dec}(e_2)_1):\mathbf{float}[B;V], & \\
\quad (\mathrm{dec}(e_1)_2, \mathrm{dec}(e_2)_2)) & \text{if } e = \mathbf{gaussian}(e_1,e_2):\mathbf{float}[B;V] \\
(\mathbf{exponential}(\mathrm{dec}(e_1)_1):\mathbf{float}[B;V], & \\
\quad \mathrm{dec}(e_1)_2) & \text{if } e = \mathbf{exponential}(e_1):\mathbf{float}[B;V] \\
(\mathbf{beta}(\mathrm{dec}(e_1)_1, \mathrm{dec}(e_2)_1):\mathbf{float}[B;V], & \\
\quad (\mathrm{dec}(e_1)_2, \mathrm{dec}(e_2)_2)) & \text{if } e = \mathbf{beta}(e_1,e_2):\mathbf{float}[B;V] \\
(\mathrm{dec}(e_1)_1 \sim \mathrm{dec}(e_2)_1 : \mathbf{bool}, & \\
\quad (\mathrm{dec}(e_1)_2, \mathrm{dec}(e_2)_2)) & \text{if } e = e_1 \sim e_2 : \mathbf{bool}, \sim \in =, \leq, <_{\#n}, \leq_{\#n} \\
(\mathbf{if}\ \mathrm{dec}(e_1)_1\ \mathbf{then}\ \mathrm{dec}(e_2)_1\ \mathbf{else}\ \mathrm{dec}(e_3)_1 : \tau, & \\
\quad (\mathrm{dec}(e_1)_2, \mathrm{dec}(e_2)_2, \mathrm{dec}(e_3)_2)) & \text{if } e = \mathbf{if}\ e_1\ \mathbf{then}\ e_2\ \mathbf{else}\ e_3 : \tau \\
(\mathbf{let}\ x = \mathrm{dec}(e_1)_1\ \mathbf{in}\ \mathrm{dec}(e_2)_1 : \tau, & \\
\quad (\mathrm{dec}(e_1)_2, \mathrm{dec}(e_2)_2)) & \text{if } e = \mathbf{let}\ x = e_1\ \mathbf{in}\ e_2 : \tau \\
(\mathbf{observe}(\mathrm{dec}(e_1)_1 : \mathbf{bool}) : \mathbf{unit}, \mathrm{dec}(e_1)_2) & \text{if } e = \mathbf{observe}(e_1 : \mathbf{bool}) : \mathbf{unit}
\end{cases}
$$

Fig. 27. Recursive definition of dec, a decomposition function, over typed expressions. $\mathrm{dec}(\cdot)_1$ and $\mathrm{dec}(\cdot)_2$ denote first and second projections of dec, respectively. $\mathrm{dec}(\cdot)_2$ is canonically bijective to a flat vector in $\mathbb{R}^n$ via coordinate concatenation over holes.

Therefore, the admissible parameter tuples are precisely

$$A_{s:\tau} = \{(c_1, c_2) \mid c_1 \in \mathbb{R},\ c_2 \in \{3.5, 4.5\}\}.$$

Equivalently,

$$A_{s:\tau} = \mathbb{R} \times \{3.5, 4.5\} \subseteq \mathbb{R}^2.$$

$$
\mathrm{dec}(e:\tau) = \begin{cases}
(\mathrm{dec}(e_1)_1 \;\&\&\; \mathrm{dec}(e_2)_1 : \textbf{bool}, \\ (\mathrm{dec}(e_1)_2, \mathrm{dec}(e_2)_2)) & \text{if } e = e_1 \;\&\&\; e_2 : \textbf{bool} \\[1em]
(\mathrm{dec}(e_1)_1 \;||\; \mathrm{dec}(e_2)_1 : \textbf{bool}, \\ (\mathrm{dec}(e_1)_2, \mathrm{dec}(e_2)_2)) & \text{if } e = e_1 \;||\; e_2 : \textbf{bool} \\[1em]
(\text{not } \mathrm{dec}(e_1)_1 : \textbf{bool}, \mathrm{dec}(e_1)_2) & \text{if } e = \text{not } e_1 : \textbf{bool} \\[1em]
(\mathrm{dec}(e_1)_1; \mathrm{dec}(e_2)_1 : \tau_2, \\ (\mathrm{dec}(e_1)_2, \mathrm{dec}(e_2)_2)) & \text{if } e = e_1; e_2 : \tau_2 \\[1em]
((\mathrm{dec}(e_1)_1, \mathrm{dec}(e_2)_1) : \tau_1 * \tau_2, \\ (\mathrm{dec}(e_1)_2, \mathrm{dec}(e_2)_2)) & \text{if } e = (e_1, e_2) : \tau_1 * \tau_2 \\[1em]
(\textbf{fst } \mathrm{dec}(e_1)_1 : \tau_1, \mathrm{dec}(e_1)_2) & \text{if } e = \textbf{fst } e_1 : \tau_1 \\[1em]
(\textbf{snd } \mathrm{dec}(e_1)_1 : \tau_2, \mathrm{dec}(e_1)_2) & \text{if } e = \textbf{snd } e_1 : \tau_2 \\[1em]
(\textbf{fun } x \to \mathrm{dec}(e_1)_1 : \tau_1 \to \tau_2, \mathrm{dec}(e_1)_2) & \text{if } e = \textbf{fun } x \to e_1 : \tau_1 \to \tau_2 \\[1em]
(\textbf{fix } f\ x := \mathrm{dec}(e_1)_1 : \tau_1 \to \tau_2, \mathrm{dec}(e_1)_2) & \text{if } e = \textbf{fix } f\ x := e_1 : \tau_1 \to \tau_2 \\[1em]
(\mathrm{dec}(e_1)_1\ \mathrm{dec}(e_2)_1 : \tau_2, \\ (\mathrm{dec}(e_1)_2, \mathrm{dec}(e_2)_2)) & \text{if } e = (e_1 : \tau_1 \to \tau_2 \;\; e_2 : \tau_1) : \tau_2 \\[1em]
(\texttt{nil} : \textbf{list}(\tau_1), ()) & \text{if } e = \texttt{nil} : \textbf{list}(\tau_1) \\[1em]
(\mathrm{dec}(e_1)_1 :: \mathrm{dec}(e_2)_1 : \textbf{list}(\tau_1), \\ (\mathrm{dec}(e_1)_2, \mathrm{dec}(e_2)_2)) & \text{if } e = e_1 :: e_2 : \textbf{list}(\tau_1) \\[1em]
(\textbf{match } \mathrm{dec}(e_0)_1 \textbf{ with } \mid \texttt{nil} \to \mathrm{dec}(e_{\text{nil}})_1 \\ \mid h :: t \to \mathrm{dec}(e_{\text{cons}})_1 \textbf{ end} : \tau_2, \\ (\mathrm{dec}(e_0)_2, \mathrm{dec}(e_{\text{nil}})_2, \mathrm{dec}(e_{\text{cons}})_2)) & \text{if } e = (\textbf{match } e_0 : \textbf{list}(\tau_1) \textbf{ with } \mid \texttt{nil} \to e_{\text{nil}} : \tau_2 \\ & \qquad \mid h :: t \to e_{\text{cons}} : \tau_2 \textbf{ end}) : \tau_2
\end{cases}
$$

Fig. 28. Recursive definition of dec over typed expressions (continued).

Thus, expressions with this syntactic shape are parameterized by a two-dimensional space whose first coordinate ranges over $\mathbb{R}$ and whose second coordinate ranges over a finite set.

Definition F.15 (σ-algebra construction on $A_{s;\tau}$). Let $s : \tau \in \mathit{Skel}_\tau$. Endow $A_{s;\tau}$ with the subspace σ-algebra inherited from the standard Borel σ-algebra on $\mathbb{R}^{holes(s)}$:

$$\mathcal{F}_{s;\tau} := \left\{ A_{s;\tau} \cap M \mid M \in \mathcal{B}(\mathbb{R}^{holes(s)}) \right\}.$$

Lemma F.16. *$(A_{s;\tau}, \mathcal{F}_{s;\tau})$ is a Borel measurable space.*

Proof. For a fixed $s : \tau \in \mathit{Skel}_\tau$, $(\mathbb{R}^{holes(s)}, \mathcal{B}(\mathbb{R}^{holes(s)}))$ is a Borel measurable space and $A_{s;\tau} \subseteq \mathbb{R}^{holes(s)}$. Thus, by Definition F.1, $(A_{s;\tau}, \{A_{s;\tau} \cap M \mid M \in \mathcal{B}(\mathbb{R}^{holes(s)})\}) = (A_{s;\tau}, \mathcal{F}_{s;\tau})$ is a Borel measurable space. □

Now, we map this measurable space to the space of actual expressions that share that skeleton.

Definition F.17 (Expressions of a skeleton). For a given $s : \tau \in Skel_\tau$, define the set of well-typed expressions having skeleton s as:

$$Expr_{s;\tau} := \{\, e : \tau \in Expr_\tau \mid \mathrm{dec}(e : \tau)_1 = s \,\} \subseteq Expr_\tau.$$

Intuitively, $Expr_{s;\tau}$ consists exactly of those expressions whose syntactic shape is fixed by s, and whose real constants vary according to the admissible hole assignments in $A_{s;\tau}$.

Example F.18. Consider the skeleton

$$s : \tau = (\square : \mathbf{float}[\{<3.5, <4.5\}; \top] \;<\; \square : \mathbf{float}[\{<3.5, <4.5\}; \{3.5, 4.5\}]) : \mathbf{bool}.$$

An expression $e : \mathbf{bool}$ belongs to $Expr_{s;\tau}$ iff it is of the form

$$e = c_1 < c_2$$

where $c_1 \in \mathbb{R}$ and $c_2 \in \{3.5, 4.5\}$.

Therefore,

$$Expr_{s;\tau} = \{c_1 < c_2 \mid c_1 \in \mathbb{R},\ c_2 \in \{3.5, 4.5\}\}\,.$$

Definition F.19 (σ-algebra construction on $Expr_{s;\tau}$). Let $s : \tau \in Skel_\tau$. Endow $Expr_{s;\tau}$ with the following σ-algebra:

$$\mathcal{F}_{Expr_{s;\tau}} := \left\{\{e : \tau \in Expr_{s;\tau} \mid \mathrm{dec}(e : \tau)_2 \in B\} \mid B \in \mathcal{F}_{s;\tau}\right\}.$$

Lemma F.20. *For a fixed $s : \tau \in Skel_\tau$, the map*

$$\mathrm{dec}_{s;\tau} : Expr_{s;\tau} \to A_{s;\tau}, \qquad e \mapsto \mathrm{dec}(e)_2.$$

is a bijection.

Proof. Let $s : \tau \in Skel_\tau$.

(i) Injectivity: Suppose $e_1, e_2 \in Expr_{s;\tau}$ satisfy $\mathrm{dec}_{s;\tau}(e_1) = \mathrm{dec}_{s;\tau}(e_2)$. By definition of $\mathrm{dec}_{s;\tau}$, $\mathrm{dec}_{s;\tau}(e_1) = \mathrm{dec}_{s;\tau}(e_2) = \vec{r}$ for some $\vec{r} \in A_{s;\tau}$. Since both e_1 and e_2 have skeleton s by assumption, we have $e_1 = s[\vec{r}]$ and $e_2 = s[\vec{r}]$, hence $e_1 = e_2$.

(ii) Surjectivity: Let $\vec{r} \in A_{s;\tau}$. By definition of $A_{s;\tau}$, there exists some expression $e \in Expr_\tau$ such that $\mathrm{dec}(e) = (s, \vec{r})$. Then $e \in Expr_{s;\tau}$ by construction of $Expr_{s;\tau}$, and $\mathrm{dec}_{s;\tau}(e) = \vec{r}$.

$\mathrm{dec}_{s;\tau}$ is injective and surjective, hence a bijection. □

Lemma F.21 (Isomorphism property). *For $s \in Skel_\tau$,*

$$(Expr_{s;\tau}, \mathcal{F}_{Expr_{s;\tau}}) \cong (A_{s;\tau}, \mathcal{F}_{s;\tau}).$$

Proof. We know $(A_{s;\tau}, \mathcal{F}_{s;\tau})$ is a Borel measurable space by Lemma F.16. First, verify that $\mathcal{F}_{Expr_{s;\tau}}$ coincides with the σ-algebra generated by $\mathrm{dec}_{s;\tau}$, i.e. $\mathcal{F}_{Expr_{s;\tau}} = \mathrm{dec}_{s;\tau}^{-1}(\mathcal{F}_{s;\tau})$:

$$\begin{aligned}
\mathcal{F}_{Expr_{s;\tau}} &:= \left\{\{e : \tau \in Expr_{s;\tau} \mid \mathrm{dec}(e : \tau)_2 \in B\} \mid B \in \mathcal{F}_{s;\tau}\right\} && \text{(Definition F.19)}\\
&:= \left\{\mathrm{dec}_{s;\tau}^{-1}(B) \mid B \in \mathcal{F}_{s;\tau}\right\} && \text{(unfolding the preimage of } \mathrm{dec}_{s;\tau})\\
&:= \mathrm{dec}_{s;\tau}^{-1}(\mathcal{F}_{s;\tau}) && \text{(Definition F.2)}
\end{aligned}$$

By Lemma F.20, $\mathrm{dec}_{s;\tau} : Expr_{s;\tau} \to A_{s;\tau}$ is a bijection. By Lemma F.3, $\mathrm{dec}_{s;\tau}$ induces an isomorphism $(Expr_{s;\tau}, \mathcal{F}_{Expr_{s;\tau}}) \cong (A_{s;\tau}, \mathcal{F}_{s;\tau})$. □

Lemma F.22. *$(Expr_{s;\tau}, \mathcal{F}_{Expr_{s;\tau}})$ is a Borel measurable space.*

Proof. By Lemma F.21. □

Now we combine the measurable spaces for each distinct skeleton into a single measurable space of all well-typed expressions of type τ. The set of all well-typed expressions of type τ is thus the disjoint union:

$$Expr_\tau = \bigsqcup_{s:\tau \in Skel_\tau} Expr_{s;\tau}.$$

This union is disjoint because every typed expression $e : \tau$ has a unique skeleton, namely $\mathrm{dec}(e)_1$, so e belongs to exactly one set $Expr_{s;\tau}$.

Definition F.23 (σ-algebra construction on $Expr_\tau$). Endow $Expr_\tau$ with the following σ-algebra:

$$\mathcal{F}_{Expr_\tau} := \{A \subseteq Expr_\tau \mid \forall s \in Skel_\tau.\ A \cap Expr_{s;\tau} \in \mathcal{F}_{Expr_{s;\tau}}\}$$

Lemma F.24. *$(Expr_\tau, \mathcal{F}_{Expr_\tau})$ is a measurable space.*

Proof. We verify that $\mathcal{F}_{Expr_\tau}$ is indeed a σ-algebra on $Expr_\tau$.

(i) Contains $Expr_\tau$: We show $Expr_\tau \in \mathcal{F}_{Expr_\tau}$. By Definition F.23, it suffices to show that $\forall s \in Skel_\tau$, $Expr_\tau \cap Expr_{s;\tau} \in \mathcal{F}_{Expr_{s;\tau}}$.

Observe that:

$$\begin{aligned} & Expr_\tau \cap Expr_{s;\tau} \\ &= Expr_{s;\tau} && \text{(since } Expr_{s;\tau} \subseteq Expr_\tau\text{)} \end{aligned}$$

Because $\mathcal{F}_{Expr_{s;\tau}}$ is a σ-algebra, it contains the set $Expr_{s;\tau}$. Hence,

$$Expr_{s;\tau} \in \mathcal{F}_{Expr_{s;\tau}}.$$

Therefore, $Expr_\tau \cap Expr_{s;\tau} \in \mathcal{F}_{Expr_{s;\tau}}$ for all $s \in Skel_\tau$. Therefore, $Expr_\tau \in \mathcal{F}_{Expr_\tau}$.

(ii) Closure under complements: Let $A \in \mathcal{F}_{Expr_\tau}$. We show that $Expr_\tau \setminus A \in \mathcal{F}_{Expr_\tau}$. By Definition F.23, it suffices to show that $\forall s \in Skel_\tau$, $(Expr_\tau \setminus A) \cap Expr_{s;\tau} \in \mathcal{F}_{Expr_{s;\tau}}$.

Fix $s \in Skel_\tau$. Observe that:

$$\begin{aligned} & (Expr_\tau \setminus A) \cap Expr_{s;\tau} \\ &= (Expr_\tau \cap Expr_{s;\tau}) \setminus (A \cap Expr_{s;\tau}) && (\cap \text{ distributes over } \setminus) \\ &= Expr_{s;\tau} \setminus (A \cap Expr_{s;\tau}). && \text{(since } Expr_{s;\tau} \subseteq Expr_\tau\text{, so } Expr_\tau \cap Expr_{s;\tau} = Expr_{s;\tau}\text{)} \end{aligned}$$

Since $A \in \mathcal{F}_{Expr_\tau}$ by assumption, by Definition F.23, we have $A \cap Expr_{s;\tau} \in \mathcal{F}_{Expr_{s;\tau}}$, and because $\mathcal{F}_{Expr_{s;\tau}}$ is a σ-algebra, it is closed under complements relative to $Expr_{s;\tau}$. Hence

$$Expr_{s;\tau} \setminus (A \cap Expr_{s;\tau}) \in \mathcal{F}_{Expr_{s;\tau}}.$$

Therefore $(Expr_\tau \setminus A) \cap Expr_{s;\tau} \in \mathcal{F}_{Expr_{s;\tau}}$ for all $s \in Skel_\tau$. Therefore, $Expr_\tau \setminus A \in \mathcal{F}_{Expr_\tau}$.

(iii) Closure under countable unions: Let $(A_n)_{n\in\mathbb{N}}$ be a sequence with $A_n \in \mathcal{F}_{Expr_\tau}$ for all n. We show that $\bigcup_{n\in\mathbb{N}} A_n \in \mathcal{F}_{Expr_\tau}$. By Definition F.23, it suffices to show that $\forall s \in Skel_\tau$, $(\bigcup_{n\in\mathbb{N}} A_n) \cap Expr_{s;\tau} \in \mathcal{F}_{Expr_{s;\tau}}$.

Fix $s \in Skel_\tau$. Using distributivity of $\cap$ over $\cup$, we have

$$\left(\bigcup_{n\in\mathbb{N}} A_n\right) \cap Expr_{s;\tau} = \bigcup_{n\in\mathbb{N}} \left(A_n \cap Expr_{s;\tau}\right).$$

Since each $A_n \in \mathcal{F}_{Expr_\tau}$ by assumption, by Definition F.23, we have $A_n \cap Expr_{s;\tau} \in \mathcal{F}_{Expr_{s;\tau}}$ for all n. Because $\mathcal{F}_{Expr_{s;\tau}}$ is a σ-algebra, it is closed under countable unions, hence

$$\bigcup_{n\in\mathbb{N}} \left(A_n \cap Expr_{s;\tau}\right) \in \mathcal{F}_{Expr_{s;\tau}}.$$

Therefore $(\bigcup_{n\in\mathbb{N}} A_n) \cap Expr_{s;\tau} \in \mathcal{F}_{Expr_{s;\tau}}$ for all s, and thus $\bigcup_{n\in\mathbb{N}} A_n \in \mathcal{F}_{Expr_\tau}$.

Since $\mathcal{F}_{Expr_\tau}$ is a σ-algebra on $Expr_\tau$, $(Expr_\tau, \mathcal{F}_{Expr_\tau})$ is a measurable space. □

Definition F.25 (Extension with error element). Let $(Expr_\tau, \mathcal{F}_{Expr_\tau})$ be the measurable space of well-typed expressions of type τ. Define

$$Expr_\tau + \bot := Expr_\tau \cup \{\bot\},$$

equipped with the σ-algebra

$$\mathcal{F}_{Expr_\tau+\bot} := \{A,\ A \cup \{\bot\} \mid A \in \mathcal{F}_{Expr_\tau}\}.$$

Lemma F.26. *$(Expr_\tau + \bot, \mathcal{F}_{Expr_\tau+\bot})$ is a measurable space.*

Proof. We verify that $\mathcal{F}_{Expr_\tau+\bot}$ is indeed a σ-algebra on $Expr_\tau + \bot$.

By construction, every element of $\mathcal{F}_{Expr_\tau+\bot}$ is of the form A or $A \cup \{\bot\}$ for some $A \in \mathcal{F}_{Expr_\tau}$.

(i) Contains $Expr_\tau + \bot$: Take $A = Expr_\tau \in \mathcal{F}_{Expr_\tau}$. Then,

$$A \cup \{\bot\} = Expr_\tau \cup \{\bot\} = Expr_\tau + \bot \in \mathcal{F}_{Expr_\tau+\bot}.$$

(ii) Closure under complements: Let $C \in \mathcal{F}_{Expr_\tau+\bot}$. There are two cases:

- If $C = A$: Then, $(Expr_\tau + \bot) \setminus A = (Expr_\tau \setminus A) \cup \{\bot\}$. Since $\mathcal{F}_{Expr_\tau}$ is a σ-algebra, $Expr_\tau \setminus A \in \mathcal{F}_{Expr_\tau}$, so $(Expr_\tau \setminus A) \cup \{\bot\}$ is of the form $A \cup \{\bot\}$ for some $A \in \mathcal{F}_{Expr_\tau}$ and hence lies in $\mathcal{F}_{Expr_\tau+\bot}$.
- If $C = A \cup \{\bot\}$: Then, $(Expr_\tau + \bot) \setminus (A \cup \{\bot\}) = Expr_\tau \setminus A$. Since $\mathcal{F}_{Expr_\tau}$ is a σ-algebra, $Expr_\tau \setminus A \in \mathcal{F}_{Expr_\tau}$, so $Expr_\tau \setminus A$ is of the form A for some $A \in \mathcal{F}_{Expr_\tau}$ and hence lies in $\mathcal{F}_{Expr_\tau+\bot}$.

In both cases the complement lies in $\mathcal{F}_{Expr_\tau+\bot}$.

(iii) Closure under countable unions: Let $(C_n)_{n\in\mathbb{N}}$ be a sequence with $C_n \in \mathcal{F}_{Expr_\tau+\bot}$ for all n. For each n, define $A_n := C_n \cap Expr_\tau \in \mathcal{F}_{Expr_\tau}$, where the latter holds since C_n is of the form A or $A \cup \{\bot\}$ for some $A \in \mathcal{F}_{Expr_\tau}$, and in either case $C_n \cap Expr_\tau = A$.

Let $A := \bigcup_{n\in\mathbb{N}} A_n$. Since $\mathcal{F}_{Expr_\tau}$ is a σ-algebra, $A \in \mathcal{F}_{Expr_\tau}$. Now observe:

$$\bigcup_{n\in\mathbb{N}} C_n = A \cup \left(\bigcup_{n\in\mathbb{N}} (C_n \setminus Expr_\tau)\right).$$

The second factor is either $\emptyset$ (if every $C_n = A_n$) or $\{\bot\}$ (if some $C_n = A_n \cup \{\bot\}$). In the first sub-case, $\bigcup_n C_n = A$ is of the form A for some $A \in \mathcal{F}_{Expr_\tau}$; in the second, $\bigcup_n C_n = A \cup \{\bot\}$ is of the form $A \cup \{\bot\}$ for some $A \in \mathcal{F}_{Expr_\tau}$. In both cases $\bigcup_n C_n \in \mathcal{F}_{Expr_\tau+\bot}$. □

Finally, we can equip this measurable space $(Expr_\tau + \bot, \mathcal{F}_{Expr_\tau+\bot})$ with a subprobability measure such that $\mu(Expr_\tau + \bot) \leq 1$.

We give some basic lemmas:

Lemma F.27. *Let $(Expr_\tau + \bot, \mathcal{F}_{Expr_\tau+\bot})$ be the Borel measurable space of well-typed expressions with error from Definition F.25. Let $Vals_\tau \subset Expr_\tau$ be the set of all well-typed value expressions. Let $N_\tau \subset Expr_\tau$, defined as $N = Expr_\tau \setminus Vals_\tau$, be the set of all well-typed non-value expressions. Let $\{\bot\}$ be the singleton set containing the error element $\bot$. The following are true:*

(1) The set $Vals_\tau \in \mathcal{F}_{Expr_\tau+\bot}$.
(2) The set $\{\bot\} \in \mathcal{F}_{Expr_\tau+\bot}$.
(3) The set $N_\tau \in \mathcal{F}_{Expr_\tau+\bot}$.

Proof. We prove the measurability of $Vals_\tau$, $\{\bot\}$, and N_τ in separate parts. For each, we show that it belongs to the σ-algebra $\mathcal{F}_{Expr_\tau+\bot}$ on the extended expression space $Expr_\tau + \bot$.

(1) **(Measurability of $Vals_\tau$):** Every expression skeleton $s \in Skel_\tau$ is either a value skeleton or a non-value skeleton, depending on whether the syntactic form of s is a value. An expression $e \in Expr_\tau$ is a value iff its skeleton is a value skeleton.
To show $Vals_\tau \in \mathcal{F}_{Expr_\tau}$, by Definition F.23, it suffices to show that $\forall s \in Skel_\tau$, $Vals_\tau \cap Expr_{s;\tau} \in \mathcal{F}_{Expr_{s;\tau}}$.
Fix $s \in Skel_\tau$:
- If s is a value skeleton, then every expression in $Expr_{s;\tau}$ is a value, so $Vals_\tau \cap Expr_{s;\tau} = Expr_{s;\tau} \in \mathcal{F}_{Expr_{s;\tau}}$.
- If s is a non-value skeleton, then no expression in $Expr_{s;\tau}$ is a value, so $Vals_\tau \cap Expr_{s;\tau} = \emptyset \in \mathcal{F}_{Expr_{s;\tau}}$.

Thus, $Vals_\tau \cap Expr_{s;\tau} \in \mathcal{F}_{Expr_{s;\tau}}$ for any $s \in Skel_\tau$, so $Vals_\tau \in \mathcal{F}_{Expr_\tau} \subseteq \mathcal{F}_{Expr_\tau+\bot}$.

(2) **(Measurability of $\{\bot\}$):** Taking $A = \emptyset$ in Definition F.25 gives $\{\bot\} \in \mathcal{F}_{Expr_\tau+\bot}$.

(3) **(Measurability of N_τ):** $N_\tau = Expr_\tau \setminus Vals_\tau$ is the complement of $Vals_\tau$ in $Expr_\tau$. Since $Vals_\tau \in \mathcal{F}_{Expr_\tau}$ and $\mathcal{F}_{Expr_\tau}$ is a σ-algebra, $N_\tau \in \mathcal{F}_{Expr_\tau} \subseteq \mathcal{F}_{Expr_\tau+\bot}$.

□

Note that in general, given (X, Σ_X) and (Y, Σ_Y) to be measurable spaces, $((X \times Y) + \bot)$ forms a measurable space, since the product $(X \times Y)$ is measurable and for any measurable space (Z, Σ_Z), the space $(Z + \bot, \Sigma_{Z+\bot})$ with $\Sigma_{Z+\bot}$ generated by $\Sigma_Z \cup \bot$ is measurable.

F.3 Distribution Monad with Error

Let X be a set equipped with a suitable measurable space so that $\mathcal{D}(X + \bot)$ denotes the set of subprobability measures over $X + \bot$. We will make heavy use of the following generic monad construction, for which one easily verifies the monad laws:

Definition F.28 (Distribution Monad with Error). We define

$$\begin{aligned}
&\mathrm{return} \colon X \to \mathcal{D}(X + \bot), \quad \mathrm{return}(x) = \delta_x \\
&\mathrm{bind} \colon \mathcal{D}(X + \bot) \to ((X + \bot) \to \mathcal{D}(Y + \bot)) \to \mathcal{D}(Y + \bot), \\
&\mathrm{bind}(\mu)(f)(A) = \int_{X+\bot} f(x)(A) \cdot \mu(dx)\,,
\end{aligned}$$

where δ_x denotes the Dirac measure for x, and f is a sub-Markov kernel.

Definition F.29 ($\bot$-extension). Define the $\bot$-extension of $g : X \to \mathcal{D}(Y + \bot)$ to be $g_\bot \colon (X + \bot) \to \mathcal{D}(Y + \bot)$, obtained from g by:

$$g_\bot(e) = \begin{cases} \delta_\bot & \text{if } e = \bot \\ g(e) & \text{otherwise}\,. \end{cases}$$

Lemma F.30 ($\bot$-extension is a sub-Markov kernel). *Given a sub-Markov kernel $f : X \to \mathcal{D}(Y + \bot)$, its $\bot$ extension $f_\bot : (X + \bot) \to \mathcal{D}(Y + \bot)$ is also a sub-Markov kernel.*

Proof. We verify the two conditions for $f_\bot$ to be a sub-Markov kernel.

(1) Subprobability measure: We show for each $e \in X + \bot$, the map

$$A \mapsto f_\bot(e)(A)$$

is a subprobability measure.

Fix $e \in X + \bot$.

- If $e = \bot$, then $f_\bot(\bot) = \delta_\bot$, the Dirac measure at $\bot$, which is a subprobability measure on $Y + \bot$.
- If $e \in X$, then $f_\bot(e) = f(e)$, and since f is a sub-Markov kernel by assumption, $f(e)$ is a subprobability measure on $Y + \bot$.

Thus, for all $e \in X + \bot$, the map $A \mapsto f_\bot(e)(A)$ is a subprobability measure.

(2) Measurable function: We show for each $A \in \mathcal{F}_{Y+\bot}$, the map $e \mapsto f_\bot(e)(A)$ is a measurable function.

Fix $A \in \mathcal{F}_{Y+\bot}$. Define

$$g\colon (X + \bot) \to [0,1], \qquad g(e) := f_\bot(e)(A).$$

We prove g is measurable by showing $g^{-1}(B) \in \mathcal{F}_{X+\bot}$ for every $B \in \mathcal{B}([0,1])$. It suffices to prove that for every $a \in \mathbb{R}$,

$$g^{-1}((a,\infty) \cap [0,1]) \in \mathcal{F}_{X+\bot}$$

because the family of sets $\{(a,\infty) \cap [0,1] : a \in \mathbb{R}\}$ generates $\mathcal{B}([0,1])$.

Because $g(e) \in [0,1]$ for all $e \in X + \bot$ by definition, we have for every $a \in \mathbb{R}$,

$$\begin{aligned} g^{-1}\big((a,\infty) \cap [0,1]\big) &= \{e \in X + \bot \mid g(e) \in (a,\infty) \cap [0,1]\} \\ &= \{e \in X + \bot \mid (g(e) \in (a,\infty)) \wedge (g(e) \in [0,1])\} \\ &= \{e \in X + \bot \mid (g(e) > a) \wedge (g(e) \in [0,1])\} \\ &= \{e \in X + \bot \mid g(e) > a\}, \end{aligned}$$

where the last step uses that $g(e) \in [0,1]$ always holds. Hence it suffices to show that $\{e \in X + \bot \mid g(e) > a\} \in \mathcal{F}_{X+\bot}$ for every $a \in \mathbb{R}$.

- If $a \geq 1$: $\{e \in X + \bot \mid g(e) > a\} = \emptyset \in \mathcal{F}_{X+\bot}$.
- If $a < 0$: $\{e \in X + \bot \mid g(e) > a\} = X + \bot \in \mathcal{F}_{X+\bot}$.
- If $0 \leq a < 1$: $\{e \in X + \bot \mid g(e) > a\}$.

The only nontrivial case is the last. It splits into two sets:

$$\begin{aligned} \{e \in X + \bot \mid g(e) > a\} &= \{e \in X + \bot \mid f_\bot(e)(A) > a\} \\ &= \{e \in X \mid f_\bot(e)(A) > a\} \cup \{\bot \mid f_\bot(e)(A) > a\} \\ &= \{e \in X \mid f(e)(A) > a\} \cup \{\bot \mid \delta_\bot(A) > a\} \end{aligned}$$

The first set is measurable in X (and hence $X + \bot$) since f is a sub-Markov kernel by assumption. The second set is either $\emptyset \in \mathcal{F}_{X+\bot}$ or $\{\bot\} \in \mathcal{F}_{X+\bot}$. Since $\mathcal{F}_{X+\bot}$ is closed under countable unions, then $g^{-1}(B) \in \mathcal{F}_{X+\bot}$ for every $B \in \mathcal{B}([0,1])$. Thus, for all $A \in \mathcal{F}_{Y+\bot}$, the map $e \mapsto f_\bot(e)(A)$ is a measurable function.

Since both conditions hold, $f_\bot$ is a sub-Markov kernel. □

Lemma F.31 (Dirac return is a sub-Markov kernel). *For any measurable space $(X, \mathcal{F}_X)$, the map*

$$\mathsf{return} : X \to \mathcal{D}(X + \bot), \qquad x \mapsto \delta_x.$$

is a sub-Markov kernel.

Proof. We verify the two conditions for return to be a sub-Markov kernel.

(1) Subprobability measure: We show for each $x \in X$, the map

$$A \mapsto \mathsf{return}(x)(A) = \delta_x(A)$$

is a subprobability measure.

Fix $x \in X$. δ_x is the Dirac measure at x, hence a probability measure and therefore a subprobability measure on $X + \bot$. Thus, for every $x \in X$, the map $A \mapsto \delta_x(A)$ is a subprobability measure.

(2) Measurable function: We show that for each $A \in \mathcal{F}_{X+\bot}$, the map

$$x \mapsto \mathsf{return}(x)(A) = \delta_x(A)$$

is a measurable function.

Fix $A \in \mathcal{F}_{X+\bot}$. Define

$$g : X \to [0,1], \qquad g(x) := \delta_x(A).$$

We prove g is measurable by showing $g^{-1}(B) \in \mathcal{F}_X$ for every $B \in \mathcal{B}([0,1])$. It suffices to prove that for every $a \in \mathbb{R}$,

$$g^{-1}\big((a,\infty) \cap [0,1]\big) \in \mathcal{F}_X,$$

because the family $\{(a,\infty) \cap [0,1] : a \in \mathbb{R}\}$ generates $\mathcal{B}([0,1])$.

Because $g(e) \in [0,1]$ for all $e \in X + \bot$, we have for every $a \in \mathbb{R}$,

$$\begin{aligned} g^{-1}\big((a,\infty) \cap [0,1]\big) &= \{e \in X \mid g(e) \in (a,\infty) \cap [0,1]\} \\ &= \{e \in X \mid (g(e) \in (a,\infty)) \wedge (g(e) \in [0,1])\} \\ &= \{e \in X \mid (g(e) > a) \wedge (g(e) \in [0,1])\} \\ &= \{e \in X \mid g(e) > a\}, \end{aligned}$$

where the last step uses that $g(e) \in [0,1]$ always holds. Hence it suffices to show that $\{e \in X \mid g(e) > a\} \in \mathcal{F}_X$ for every $a \in \mathbb{R}$.

- If $a \geq 1$: $\{x \in X \mid g(x) > a\} = \emptyset \in \mathcal{F}_X$.
- If $a < 0$: $\{x \in X \mid g(x) > a\} = X \in \mathcal{F}_X$.
- If $0 \leq a < 1$: $\{x \in X \mid g(x) > a\}$.

The only nontrivial case is the last. Using $g(x) = \delta_x(A)$ and the fact that $\delta_x(A) \in \{0,1\}$ for all $x \in X$, we have for $0 \leq a < 1$:

$$\begin{aligned} \{x \in X \mid g(x) > a\} &= \{x \in X \mid \delta_x(A) > a\} \\ &= \{x \in X \mid \delta_x(A) = 1\} \\ &= \{x \in X \mid x \in A\} \\ &= A \cap X. \end{aligned}$$

Since $A \in \mathcal{F}_{X+\bot}$ and $\mathcal{F}_X = \{X \cap A \mid A \in \mathcal{F}_{X+\bot}\}$ is the subspace σ-algebra on $X \subseteq X + \bot$, it follows that $A \cap X \in \mathcal{F}_X$.

Therefore $\{x \in X \mid g(x) > a\} \in \mathcal{F}_X$ for all $a \in \mathbb{R}$, so $g^{-1}\big((a,\infty) \cap [0,1]\big) \in \mathcal{F}_X$ for all a. Hence $g^{-1}(B) \in \mathcal{F}_X$ for every $B \in \mathcal{B}([0,1])$, and g is measurable. □

Lemma F.32 (Zero kernel is a sub-Markov kernel). *Let $(X, \mathcal{F}_X)$ and $(Y, \mathcal{F}_Y)$ be measurable spaces. The constant-zero map*

$$f : X \to \mathcal{D}(Y), \qquad f(x)(A) := 0$$

is a sub-Markov kernel.

Proof. We verify the two kernel conditions.

(1) Subprobability measure. We show for each $x \in X$, the map

$$A \mapsto f(x)(A)$$

is a subprobability measure.

Fix any $x \in X$. Then $f(x)$ is the zero measure: it satisfies σ-additivity since

$$f(x)\left(\bigcup_i A_i\right) = 0 = \sum_i 0 = \sum_i f(x)(A_i)$$

for every pairwise disjoint family $(A_i)_i$, and total mass $f(x)(Y) = 0 \leq 1$.

(2) Measurable function. For any $A \in \mathcal{F}_Y$, the map $x \mapsto f(x)(A) = 0$ is the constant-zero function on X, hence measurable.

Thus both conditions hold, so f is a sub-Markov kernel. □

Lemma F.33 (Composition of sub-Markov kernels). *Let (X, Σ_X), (Y, Σ_Y), and (Z, Σ_Z) be measurable spaces. If*

$$P : X \to \mathcal{D}(Y) \quad \text{and} \quad Q : Y \to \mathcal{D}(Z)$$

are sub-Markov kernels, then their composition

$$(P \gg\!= Q)(x, A) \;:=\; \int_Y Q(y, A)\, P(x, dy)$$

defines a sub-Markov kernel

$$P \gg\!= Q : X \to \mathcal{D}(Z).$$

Proof. For each $x \in X$, the map $A \mapsto (P \gg\!= Q)(x, A)$ is a measure by the standard integral construction. Its total mass is bounded by

$$(P \gg\!= Q)(x, Z) = \int_Y Q(y, Z)\, P(x, dy) \leq \int_Y 1\, P(x, dy) = P(x, Y) \leq 1.$$

For each $A \in \Sigma_Z$, measurability of $y \mapsto Q(y, A)$ and the standard measurability theorem for integration against a kernel imply that $x \mapsto \int_Y Q(y, A)\, P(x, dy)$ is measurable; see [30, Lemma 1.38]. Hence the composition is a sub-Markov kernel. □

F.4 Small-Step Semantics

Throughout this section, we denote by $e : \tau$ a well-typed expression of type τ and implicitly assume that all sub-expressions are annotated with types, as well. Moreover, for the typed expression $e : \tau = \mathbf{uniform}(e_1 : \mathbf{float}[B_1; V_1], e_2 : \mathbf{float}[B_2; V_2]) : \mathbf{float}[B, V]$ with $B, B_1, B_2 \neq \top$ and $V_1, V_2 \neq \top$, we introduce an auxiliary statement $\mathtt{DC}_{B,B_1,B_2,V_1,V_2}$ and let

$$\mathcal{D}[\![e : \tau]\!] = \mathtt{DC}_{B,B_1,B_2,V_1,V_2}(\mathcal{D}[\![e_1 : \mathbf{float}[B_1; V_1]]\!], \mathcal{D}[\![e_2 : \mathbf{float}[B_2; V_2]]\!])\ .$$

We define the semantics of $\mathtt{DC}_{B,B_1,B_2,V_1,V_2}$ in such a way that it behaves like the discretized program in the last case of Figure 13 but resolves the let-bindings and the conditional choices in a single step, which will simplify our proof. We equip this statement with the following typing rule :

$$\frac{e_1 : \mathbf{fin}(|B_1| + 1) \qquad e_2 : \mathbf{fin}(|B_2| + 1) \qquad \mathsf{recovers}(B_1, V_1) \qquad \mathsf{recovers}(B_2, V_2)}{\mathtt{DC}_{B,B_1,B_2,V_1,V_2}(e_1, e_2) : \mathbf{fin}(|B| + 1)}\ \textsc{TDiscreteCase}$$

Equipped with these notions, we define the small-step semantics of our language:

Definition F.34. The small-step semantics $[\![e : \tau]\!] \in \mathcal{D}(Expr_\tau + \bot)$ is defined recursively on the structure of $e : \tau$ by the rules in Table 1 and Table 2.

Table 1. Small-step operational semantics. In the last case, the existence of suitable v_i and w_j is guaranteed since recovers(B_i, V_i) and $e_i \in V_i$ for $i \in \{1, 2\}$. Here, every function of the form $f(g) = \lambda g.\delta_{\dots}$ is extended to $f_\perp$ in the sense of Definition F.29.

$\mathbf{e} : \boldsymbol{\tau}$	**constraints**	$[\![\boldsymbol{e} : \boldsymbol{\tau}]\!]$
$\mathtt{diverge} : \tau$		$\lambda A.0$
$v : \tau$		$\delta_{v:\tau}$
$\mathtt{let}\ x = e_1 : \tau_1\ \mathtt{in}\ e_2 : \tau_2 : \tau_2$	$e_1 = v$ e_1 no val.	$\delta_{e_2[v/x]:\tau_2}$ $[\![e_1 : \tau_1]\!] \gg\!= \lambda g.\delta_{\mathtt{let}\ x=g\ \mathtt{in}\ e_2:\tau_2:\tau_2}$
$\mathtt{if}\ e_1 : \mathtt{bool}\ \mathtt{then}\ e_2 : \tau\ \mathtt{else}\ e_3 : \tau : \tau$	$e_1 = \text{true}$ $e_1 = \text{false}$ o.w.	$\delta_{e_2:\tau}$ $\delta_{e_3:\tau}$ $[\![e_1 : \mathtt{bool}]\!] \gg\!= \lambda g.\delta_{\mathtt{if}\ g\ \mathtt{then}\ e_2:\tau\ \mathtt{else}\ e_3:\tau:\tau}$
$\mathtt{observe}\ (e : \mathtt{bool}) : \mathtt{unit}$	$e = \text{true}$ $e = \text{false}$ o.w.	$\delta_{\mathtt{unit}}$ $\delta_\perp$ $[\![e : \mathtt{bool}]\!] \gg\!= \lambda g.\delta_{\mathtt{observe}\ (g:\mathtt{bool}):\mathtt{unit}}$
$e_1 : \tau_1 < e_2 : \tau_2 : \mathtt{bool}$ for $\tau_1 = \mathtt{float}[B; V_1], \tau_2 = \mathtt{float}[B; V_2]$ or $\tau_1 = \tau_2 = \mathtt{fin}(n)$	$e_1 = v_1, e_2 = v_2, v_1 < v_2$ $e_1 = v_1, e_2 = v_2, v_1 \geq v_2$ $e_1 = v_1, e_2 \neq v_2$ o.w.	$\delta_{\text{true}:\mathtt{bool}}$ $\delta_{\text{false}:\mathtt{bool}}$ $[\![e_2 : \tau_2]\!] \gg\!= \lambda g.\delta_{v_1:\tau_1<g:\mathtt{bool}}$ $[\![e_1 : \tau_1]\!] \gg\!= \lambda g.\delta_{g<e_2:\tau_2:\mathtt{bool}}$
$\mathtt{uniform}(e_1 : \tau_1, e_2 : \tau_2) : \mathtt{float}[B; V]$	$e_1 = v_1, e_2 = v_2, v_1 < v_2$ $e_1 = v_1, e_2 = v_2, v_1 \geq v_2$ $e_1 = v_1, e_2 \neq v_2$ o.w.	$\text{Uniform}(v_1, v_2) \gg\!= \lambda v.\delta_{(v:\mathtt{float}[B;V])}$ $[\![\mathtt{diverge} : \tau]\!]$ $[\![e_2 : \tau_2]\!] \gg\!= \lambda g.\delta_{\mathtt{uniform}(v_1:\tau_1,g):\tau}$ $[\![e_1 : \tau_1]\!] \gg\!= \lambda g.\delta_{\mathtt{uniform}(g,e_2:\tau_2):\tau}$
$\mathtt{discrete}(v_0, \dots, v_{n-1}) : \mathtt{fin}(n)$		$\text{Discrete}(v_0, \dots, v_{n-1})$
$\mathtt{DC}_{B,B_1,B_2,V_1,V_2}(e_1 : \tau_1, e_2 : \tau_2) : \mathtt{fin}(\lvert B\rvert + 1)$ $\tau = \mathtt{fin}(\lvert B\rvert + 1)$, $\tau_1 = \mathtt{fin}(\lvert B_1\rvert + 1), \tau_2 = \mathtt{fin}(\lvert B_2\rvert + 1)$, $V_1 = \{v_1, \dots, v_n\}, V_2 = \{w_1, \dots, w_m\}$	$e_1 = v',\ e_2 = w'$, $V_1 = \emptyset \vee V_2 = \emptyset$ $e_1 = \mathcal{D}[\![v_i : \mathtt{float}[B_1; V_1]]\!]$, $e_2 = \mathcal{D}[\![w_j : \mathtt{float}[B_2; V_2]]\!]$ $e_1 = v', e_2 \neq w'$ o.w.	$[\![\mathtt{diverge} : \tau]\!]$ $[\![\mathcal{D}[\![\mathtt{uniform}(v_i, w_j) : \mathtt{float}[B; \top]]\!] : \tau]\!]$ $[\![e_2 : \tau_2]\!] \gg\!= \lambda g.\delta_{\mathtt{DC}_{B,B_1,B_2,V_1,V_2}(v':\tau_1,g)}$ $[\![e_1 : \tau_1]\!] \gg\!= \lambda g.\delta_{\mathtt{DC}_{B,B_1,B_2,V_1,V_2}(g,e_2:\tau_2)}$

The missing probability mass in $[\![e : \tau]\!]$ is the probability of encountering an error due to ill-formed parameters fed into distributions (e.g., $\mathtt{uniform}(0.8, 0.1)$). $[\![e : \tau]\!](\perp)$ is the probability of encountering an observation failure.

Lemma F.35 (Measurability of constant extension map). *Let $A \subseteq \mathbb{R}^n$ be equipped with the subspace σ-algebra*

$$\mathcal{F}_A := \{A \cap M \mid M \in \mathcal{B}(\mathbb{R}^n)\},$$

and let $B \subseteq \mathbb{R}^d$ be equipped with the subspace σ-algebra

$$\mathcal{F}_B := \{B \cap N \mid N \in \mathcal{B}(\mathbb{R}^d)\}.$$

Fix constant vectors $c_1 \in \mathbb{R}^{m_1}, \dots, c_k \in \mathbb{R}^{m_k}$. Define

$$g : \mathbb{R}^n \to \mathbb{R}^d$$

to be any map obtained by concatenating blocks consisting of

- *copies of the input vector $\vec{r} \in \mathbb{R}^n$, and*
- *the constant vectors $c_1, \dots, c_k$,*

Table 2. Small-step operational semantics (continued). Here, every function of the form $f(g) = \lambda g.\delta_{\dots}$ is extended to $f_\perp$ in the sense of Definition F.29.

$\mathbf{e} : \boldsymbol{\tau}$	**constraints**	$[\![e : \tau]\!]$
$e_1 : \tau_1$ && $e_2 : \tau_2$: **bool** $\tau_1 =$ **bool**, $\tau_2 =$ **bool**	$e_1 = b_1 = \text{true}, e_2 = b_2 = \text{true}$ $e_1 = b_1, e_2 = b_2$ $e_1 = b_1, e_2 \neq b_2$ o.w.	$\delta_{\text{true}:\mathbf{bool}}$ $\delta_{\text{false}:\mathbf{bool}}$ $[\![e_2 : \tau_2]\!] \gg\!= \lambda g.\delta_{b_1:\tau_1 \&\& g:\mathbf{bool}}$ $[\![e_1 : \tau_1]\!] \gg\!= \lambda g.\delta_{g \&\& e_2:\tau_2:\mathbf{bool}}$
$e_1 : \tau_1$ || $e_2 : \tau_2$: **bool** $\tau_1 =$ **bool**, $\tau_2 =$ **bool**	$e_1 = b_1 = \text{false}, e_2 = b_2 = \text{false}$ $e_1 = b_1, e_2 = b_2$ $e_1 = b_1, e_2 \neq b_2$ o.w.	$\delta_{\text{false}:\mathbf{bool}}$ $\delta_{\text{true}:\mathbf{bool}}$ $[\![e_2 : \tau_2]\!] \gg\!= \lambda g.\delta_{b_1:\tau_1 \,\|\| g:\mathbf{bool}}$ $[\![e_1 : \tau_1]\!] \gg\!= \lambda g.\delta_{g \,\|\| e_2:\tau_2:\mathbf{bool}}$
not $e : \tau$: **bool** $\tau =$ **bool**	$e = \text{true}$ $e = \text{false}$ o.w.	$\delta_{\text{false}:\mathbf{bool}}$ $\delta_{\text{true}:\mathbf{bool}}$ $[\![e : \tau]\!] \gg\!= \lambda g.\delta_{\text{not } g:\mathbf{bool}}$
$(e_1 : \tau_1, e_2 : \tau_2) : \tau_1 * \tau_2$	$e_1 = v_1, e_2 = v_2$ $e_1 = v_1, e_2 \neq v_2$ o.w.	$\delta_{(v_1,v_2):\tau_1*\tau_2}$ $[\![e_2 : \tau_2]\!] \gg\!= \lambda g.\delta_{(v_1:\tau_1,f):\tau_1*\tau_2}$ $[\![e_1 : \tau_1]\!] \gg\!= \lambda g.\delta_{(f,e_2:\tau_2):\tau_1*\tau_2}$
fst $(e : \tau_1 * \tau_2) : \tau_1$	$e = (v_1, v_2)$ o.w.	$\delta_{v_1:\tau_1}$ $[\![e : \tau_1 * \tau_2]\!] \gg\!= \lambda g.\delta_{\mathbf{fst}\ e:\tau_1}$
snd $(e : \tau_1 * \tau_2) : \tau_2$	$e = (v_1, v_2)$ o.w.	$\delta_{v_2:\tau_2}$ $[\![e : \tau_1 * \tau_2]\!] \gg\!= \lambda g.\delta_{\mathbf{snd}\ e:\tau_2}$
$(e_1 : \tau_1 \to \tau_2\ e_2 : \tau_1) : \tau_2$	$e_1 =$ **fun** $x \to e$ or **fix** $g\ x := e, e_2 = v_2$ $e_1 =$ **fun** $x \to e$ or **fix** $g\ x := e, e_2 \neq v_2$ o.w.	$\delta_{e[v_2/x]:\tau_2}$ $[\![e_2 : \tau_2]\!] \gg\!= \lambda g.\delta_{e_1:\tau_1\to\tau_2\ g:\tau_2}$ $[\![e_1 : \tau_1 \to \tau_2]\!] \gg\!= \lambda g.\delta_{f\ e_2:\tau_1:\tau_2}$
nil : **list**(τ)		$\delta_{\text{nil}:\mathbf{list}(\tau)}$
$(e_1 : \tau_1 :: e_2 :$ **list**$(\tau)) :$ **list**(τ)	$e_1 = v_1, e_2 = v_2$ $e_1 = v_1, e_2 \neq v_2$ o.w.	$\delta_{v_1::v_2:\mathbf{list}(\tau)}$ $[\![e_2 : \mathbf{list}(\tau)]\!] \gg\!= \lambda g.\delta_{v_1:\tau_1::g:\mathbf{list}(\tau)}$ $[\![e_1 : \tau_1]\!] \gg\!= \lambda g.\delta_{f::e_2:\mathbf{list}(\tau):\mathbf{list}(\tau)}$
(**match** e : **list**(τ_1) **with** | nil $\to e_1 : \tau_2$ | $h :: t \to e_2 : \tau_2$ **end**) : τ_2	$e = \text{nil}$ $e = v_h :: v_t$ o.w.	$\delta_{e_1:\tau_2}$ $\delta_{e_2[v_h/h,v_t/t]:\tau_2}$ $[\![e : \mathbf{list}(\tau_1)]\!] \gg\!= \lambda g.\delta_{\mathbf{match}\ g\ \mathbf{with}\dots:\tau_2}$

in some fixed order. That is, $g(\vec{r})$ is formed by inserting the vectors $\vec{r}$ and c_j into a fixed sequence of blocks.

Assume $g(A) \subseteq B$ and define $\phi := g|_A : A \to B$. Then ϕ is measurable.

Example F.36 (Constant extension maps). The constant extension map allows constants to be inserted and the input vector $\vec{r} \in \mathbb{R}^n$ to appear multiple times. The following are examples of such maps:

- $g : \mathbb{R}^n \to \mathbb{R}^n, \quad g(\vec{r}) = \vec{r}$
- $g : \mathbb{R}^n \to \mathbb{R}^{n+m_1}, \quad g(\vec{r}) = (\vec{r}, c_1), \quad c_1 \in \mathbb{R}^{m_1}$
- $g : \mathbb{R}^n \to \mathbb{R}^{m_1+n+m_2}, \quad g(\vec{r}) = (c_1, \vec{r}, c_2), \quad c_1 \in \mathbb{R}^{m_1},\ c_2 \in \mathbb{R}^{m_2}$
- $g : \mathbb{R}^n \to \mathbb{R}^{n+m_2}, \quad g(\vec{r}) = (\vec{r}, c_2), \quad c_2 \in \mathbb{R}^{m_2}$
- $g : \mathbb{R}^n \to \mathbb{R}^{m_1+n+m_2+n+m_3+n}, \quad g(\vec{r}) = (c_1, \vec{r}, c_2, \vec{r}, c_3, \vec{r}), \quad c_i \in \mathbb{R}^{m_i}$

Proof. We show that $\phi^{-1}(S) \in \mathcal{F}_A$ for each $S \in \mathcal{F}_B$.

Fix $S \in \mathcal{F}_B$. By definition of $\mathcal{F}_B$, there exists $N \in \mathcal{B}(\mathbb{R}^d)$ such that $S = B \cap N$. We compute:

$$\begin{aligned}
\phi^{-1}(S) &= \{\vec{r} \in A \mid \phi(\vec{r}) \in B \cap N\} \\
&= \{\vec{r} \in A \mid \phi(\vec{r}) \in B \cap \phi(\vec{r}) \in N\} && \text{(definition of } \cap) \\
&= \{\vec{r} \in A \mid \phi(\vec{r}) \in N\} && (\phi(\vec{r}) \in B \text{ for all } \vec{r} \in A) \\
&= \{\vec{r} \in A \mid g(\vec{r}) \in N\} && \text{(definition of } \phi) \\
&= \{\vec{r} \in A \mid g(\vec{r}) \in N\} \cap \mathbb{R}^n && (\vec{r} \text{ ranges over } A \subseteq \mathbb{R}^n) \\
&= A \cap \{\vec{r} \in \mathbb{R}^n \mid g(\vec{r}) \in N\}. && \text{(rewrite as } \cap \text{ with } A)
\end{aligned}$$

It remains to show that $\{\vec{r} \in \mathbb{R}^n \mid (\vec{r}, \vec{c}) \in N\} \in \mathcal{B}(\mathbb{R}^n)$. Define the total constant-extension map

$$g \colon \mathbb{R}^n \to \mathbb{R}^d.$$

We claim g is continuous. A map into a product space is continuous if and only if each coordinate function is continuous [36, Theorem 19.6]. The coordinate functions of g are:

- the identity $\mathrm{id} \colon \mathbb{R}^n \to \mathbb{R}^n$, $\vec{r} \mapsto \vec{r}$, and
- the constant map $\kappa_{\vec{c}_i} \colon \mathbb{R}^n \to \mathbb{R}^{m_i}$, $\vec{r} \mapsto \vec{c}_i$.

Both are continuous: for any open $U \subseteq \mathbb{R}^n$, $\mathrm{id}^{-1}(U) = U$ is open; and for any open $V \subseteq \mathbb{R}^{m_i}$,

$$\kappa_{\vec{c}_i}^{-1}(V) = \begin{cases} \mathbb{R}^n & \text{if } \vec{c}_i \in V, \\ \emptyset & \text{if } \vec{c}_i \notin V, \end{cases}$$

both of which are open. Hence g is continuous, and therefore Borel measurable. Applying this to $N \in \mathcal{B}(\mathbb{R}^d)$:

$$\{\vec{r} \in \mathbb{R}^n \mid g(\vec{r}) \in N\} = g^{-1}(N) \in \mathcal{B}(\mathbb{R}^n).$$

Hence $\phi_{\vec{c}}^{-1}(S) = A \cap g^{-1}(N) \in \mathcal{F}_A$ by definition, so $\phi_{\vec{c}}$ is measurable. □

Lemma F.37 (Substitution is measurable). *Fix a typed expression $e : \tau_2$ and a variable $x : \tau_1$. The substitution map*

$$\mathrm{Sub}_{e,x} \colon Val_{\tau_1} \to Expr_{\tau_2}, \qquad \mathrm{Sub}_{e,x}(v) := e[v/x],$$

is measurable, where $Val_{\tau_1} \subseteq Expr_{\tau_1}$ carries the subspace σ-algebra inherited from $(Expr_{\tau_1}, \mathcal{F}_{Expr_{\tau_1}})$.

Proof. We show that $\mathrm{Sub}_{e,x}^{-1}(A) \in \mathcal{F}_{Val_{\tau_1}}$ for all $A \in \mathcal{F}_{Expr_{\tau_2}}$.

Fix $A \in \mathcal{F}_{Expr_{\tau_2}}$. We know:

$$\begin{aligned}
\mathrm{Sub}_{e,x}^{-1}(A) &= \{v \in Val_{\tau_1} \mid e[v/x] \in A\} \\
&= \{v \in Val_{\tau_1} \mid e[v/x] \in A\} \cap Expr_{\tau_1} && (v \text{ ranges over } Val_{\tau_1} \subseteq Expr_{\tau_1}) \\
&= Val_{\tau_1} \cap \{u \in Expr_{\tau_1} \mid e[u/x] \in A\} && \text{(rewrite as } \cap \text{ with } Val_{\tau_1})
\end{aligned}$$

Now, define:

$$C := \{u \in Expr_{\tau_1} \mid e[u/x] \in A\}.$$

We claim proving $C \in \mathcal{F}_{Expr_{\tau_1}}$ is sufficient because by Definition F.1, this gives $\mathrm{Sub}_{e,x}^{-1}(A) = Val_{\tau_1} \cap C \in \mathcal{F}_{Val_{\tau_1}}$. To show $C \in \mathcal{F}_{Expr_{\tau_1}}$, by Definition F.23, it suffices to show that $\forall s \in Skel_{\tau_1}, C \cap Expr_{s;\tau_1} \in \mathcal{F}_{Expr_{s;\tau_1}}$.

Fix $s \in Skel_{\tau_1}$. We must prove:

$$\begin{aligned}
& C \cap Expr_{s;\tau_1} \in \mathcal{F}_{Expr_{s;\tau_1}} \\
\iff & \{u \in Expr_{\tau_1} \mid e[u/x] \in A\} \cap Expr_{s;\tau_1} \in \mathcal{F}_{Expr_{s;\tau_1}} && \text{(definition of } C\text{)} \\
\iff & \{u \in Expr_{s;\tau_1} \mid e[u/x] \in A\} \in \mathcal{F}_{Expr_{s;\tau_1}} && (Expr_{s;\tau_1} \subseteq Expr_{\tau_1}) \\
\iff & \{s[\vec{r}] \in Expr_{s;\tau_1} \mid e[s[\vec{r}]/x] \in A\} \in \mathcal{F}_{Expr_{s;\tau_1}} && \text{(rewrite } u\text{)} \\
\iff & \{\vec{r} \in A_{s;\tau_1} \mid e[s[\vec{r}]/x] \in A\} \in \mathcal{F}_{s;\tau_1} && \text{(Lemma F.21)}
\end{aligned}$$

Define

$$R_s := \{\vec{r} \in A_{s;\tau_1} \mid e[s[\vec{r}]/x] \in A\}.$$

Let $t \in Skel_{\tau_2}$ be the skeleton obtained from e by replacing each occurrence of x with s. Define a constant extension map:

$$\psi_s : A_{s;\tau_1} \to A_{t;\tau_2}, \qquad \psi_s(\vec{r}) = \phi(\vec{r}, \vec{c_1}, ..., \vec{c_k})$$

where $\vec{c_1}, ..., \vec{c_k} \in \mathbb{R}$ are the float constants appearing in e (other than at occurrences of x), and ϕ permutes/interleaves the blocks into the order dictated by a left-to-right traversal of the holes of t.

More precisely, since t is obtained from e by replacing each free occurrence of x with s, the holes of t consist exactly of:

- the holes of s, repeated once per occurrence of x in e, and
- the holes coming from the float constants of e itself.

By Lemma F.35, ψ_s is measurable.

By construction, for all $\vec{r} \in A_{s;\tau_1}$:

$$e[s[\vec{r}]/x] = t[\psi_s(\vec{r})].$$

Define the measurable set of hole-assignments corresponding to A at skeleton t:

$$\begin{aligned}
A_t :=& \{\vec{u} \in A_{t;\tau_2} \mid t[\vec{u}] \in A\} \\
:=& \{\vec{u} \in A_{t;\tau_2} \mid t[\vec{u}] \in A \cap Expr_{t;\tau_2}\} && (t[\vec{u}] \text{ ranges over } Expr_{t;\tau_2})
\end{aligned}$$

By Definition F.23, $A \cap Expr_{t;\tau_2} \in \mathcal{F}_{Expr_{t;\tau_2}}$, and by Lemma F.3, $A_t \in \mathcal{F}_{t;\tau_2}$. We compute:

$$\begin{aligned}
R_s &= \{\vec{r} \in A_{s;\tau_1} \mid e[s[\vec{r}]/x] \in A\} \\
&= \{\vec{r} \in A_{s;\tau_1} \mid t[\psi_s(\vec{r})] \in A\} \\
&= \{\vec{r} \in A_{s;\tau_1} \mid \psi_s(\vec{r}) \in A_t\} \\
&= \psi_s^{-1}(A_t).
\end{aligned}$$

Since ψ_s is measurable and $A_t \in \mathcal{F}_{t;\tau_2}$, we have $R_s = \psi_s^{-1}(A_t) \in \mathcal{F}_{s;\tau_1}$.

Since $R_s \in \mathcal{F}_{s;\tau_1}$ for every $s \in Skel_{\tau_1}$, we get $C \in \mathcal{F}_{Expr_{\tau_1}}$, and therefore

$$\mathrm{Sub}_{e,x}^{-1}(A) = C \cap Val_{\tau_1} \in \mathcal{F}_{Val_{\tau_1}}.$$

Hence $\mathrm{Sub}_{e,x}$ is measurable. □

Lemma F.38 (Monadic bind is a sub-Markov kernel). *For every occurrence of a monadic bind $\mu \gg\!= f$ given in the small-step operational semantics in Tables 1 and 2, $f : (Expr_{\tau_1} + \bot) \to \mathcal{D}(Expr_{\tau_2} + \bot)$ is a sub-Markov kernel.*

Proof. We first prove that $f : Expr_{\tau_1} \to \mathcal{D}(Expr_{\tau_2} + \bot)$ is a sub-Markov kernel. Let e be an expression whose evaluation rule contains a monadic bind $\mu \gg\!= f$. We proceed by case analysis on e:

- Consider $e = (\textbf{let}\ y = x : \tau_1\ \textbf{in}\ e_2 : \tau_2) : \tau_2$.
  This means proving that $\lambda x.\, \delta_{\textbf{let}\ y=x\ \textbf{in}\ e_2}$ is a sub-Markov kernel. Formally, define

$$f \colon Expr_{\tau_1} \to \mathcal{D}(Expr_{\tau_2} + \bot), \qquad f(x) = \delta_{\textbf{let}\ y=x\ \textbf{in}\ e_2}.$$

  We verify the two conditions for f to be a sub-Markov kernel.

  (i) Subprobability measure: We show for each $x \in Expr_{\tau_1}$, the map $A \mapsto f(x)(A)$ is a subprobability measure.

  Fix $x \in Expr_{\tau_1}$. The map $A \mapsto f(x)(A)$ equals $A \mapsto \delta_{\textbf{let}\ y=x\ \textbf{in}\ e_2}(A)$, which satisfies:

$$f(x)(A) = \begin{cases} 1 & \text{if } \textbf{let}\ y = x\ \textbf{in}\ e_2 \in A, \\ 0 & \text{otherwise.} \end{cases}$$

  A Dirac measure is a subprobability measure, so this condition holds.

  (ii) Measurable function: We show for each $A \in \mathcal{F}_{Expr_{\tau_2}+\bot}$, the map $x \mapsto f(x)(A)$ is a measurable function.

  Fix $A \in \mathcal{F}_{Expr_{\tau_2}+\bot}$. Define

$$g \colon Expr_{\tau_1} \to [0,1], \qquad g(x) := f(x)(A) = \delta_{\textbf{let}\ y=x\ \textbf{in}\ e_2}(A).$$

  We prove g is measurable by showing $g^{-1}(B) \in \mathcal{F}_{Expr_{\tau_1}}$ for every $B \in \mathcal{B}([0,1])$. It suffices to prove that for every $a \in \mathbb{R}$,

$$g^{-1}\big((a,\infty) \cap [0,1]\big) \in \mathcal{F}_{Expr_{\tau_1}},$$

  because the family $\{(a,\infty) \cap [0,1] : a \in \mathbb{R}\}$ generates $\mathcal{B}([0,1])$.
  Because $g(x) \in [0,1]$ for all $x \in Expr_{\tau_1}$, we have for every $a \in \mathbb{R}$,

$$\begin{aligned} g^{-1}\big((a,\infty) \cap [0,1]\big) &= \{x \in Expr_{\tau_1} \mid g(x) \in (a,\infty) \cap [0,1]\} \\ &= \{x \in Expr_{\tau_1} \mid (g(x) \in (a,\infty)) \wedge (g(x) \in [0,1])\} \\ &= \{x \in Expr_{\tau_1} \mid (g(x) > a) \wedge (g(x) \in [0,1])\} \\ &= \{x \in Expr_{\tau_1} \mid g(x) > a\}, \end{aligned}$$

  where the last step uses that $g(x) \in [0,1]$ always holds. Hence it suffices to show that $\{x \in X \mid g(x) > a\} \in \mathcal{F}_{Expr_{\tau_1}}$ for every $a \in \mathbb{R}$.
  – If $a \geq 1$: $\{x \in Expr_{\tau_1} \mid g(x) > a\} = \emptyset \in \mathcal{F}_{Expr_{\tau_1}}$.
  – If $a < 0$: $\{x \in Expr_{\tau_1} \mid g(x) > a\} = Expr_{\tau_1} \in \mathcal{F}_{Expr_{\tau_1}}$.
  – If $0 \leq a < 1$: $\{x \in Expr_{\tau_1} \mid g(x) > a\}$.
  The only nontrivial case is the last. Using $g(x) = \delta_{\textbf{let}\ y=x\ \textbf{in}\ e_2}$ and the fact that $g(x) \in \{0,1\}$ for all $x \in Expr_{\tau_1}$, we have for $0 \leq a < 1$:

$$\{x \in Expr_{\tau_1} \mid g(x) > a\} = \{x \in Expr_{\tau_1} \mid g(x) = 1\}$$

  Define

$$B := \{x \in Expr_{\tau_1} \mid \textbf{let}\ y = x\ \textbf{in}\ e_2 \in A\}.$$

  We must show $B \in \mathcal{F}_{Expr_{\tau_1}}$. By Definition F.23, it suffices to show that $\forall s \in Skel_{\tau_1},\ B \cap Expr_{s;\tau_1} \in \mathcal{F}_{Expr_{s;\tau_1}}$.

Fix $s \in Skel_{\tau_1}$. We must prove:

$$\begin{aligned}
& B \cap Expr_{s;\tau_1} \in \mathcal{F}_{Expr_{s;\tau_1}} \\
\iff\ & \{x \in Expr_{\tau_1} \mid \mathbf{let}\ y = x\ \mathbf{in}\ e_2 \in A\} \cap Expr_{s;\tau_1} \in \mathcal{F}_{Expr_{s;\tau_1}} && \text{(Definition of } B\text{)} \\
\iff\ & \{x \in Expr_{s;\tau_1} \mid \mathbf{let}\ y = x\ \mathbf{in}\ e_2 \in A\} \in \mathcal{F}_{Expr_{s;\tau_1}} && (Expr_{s;\tau_1} \subseteq Expr_{\tau_1}) \\
\iff\ & \{s[\vec{r}] \in Expr_{s;\tau_1} \mid \mathbf{let}\ y = s[\vec{r}]\ \mathbf{in}\ e_2 \in A\} \in \mathcal{F}_{Expr_{s;\tau_1}} && \text{(rewrite } x\text{)} \\
\iff\ & \{\vec{r} \in A_{s;\tau_1} \mid \mathbf{let}\ y = s[\vec{r}]\ \mathbf{in}\ e_2 \in A\} \in \mathcal{F}_{s;\tau_1} && \text{(Lemma F.21)}
\end{aligned}$$

Define

$$R_s := \{\vec{r} \in A_{s;\tau_1} \mid \mathbf{let}\ y = s[\vec{r}]\ \mathbf{in}\ e_2 \in A\}.$$

Let s_2 be the skeleton of e_2, with $m := holes(s_2)$, and let $\vec{c} \in A_{s_2;\tau_2} \subseteq \mathbb{R}^m$ be the unique tuple such that $e_2 = s_2[\vec{c}]$. Define the combined skeleton

$$s' := \mathbf{let}\ y = s\ \mathbf{in}\ s_2 \in Skel_{\tau_2},$$

whose holes consist of those from s (filled by $\vec{r} \in \mathbb{R}^n$) followed by those from s_2 (filled by $\vec{c} \in \mathbb{R}^m$), so that $A_{s';\tau_2} \subseteq \mathbb{R}^{n+m}$. Thus, for every $\vec{r} \in A_{s;\tau_1}$:

$$\mathbf{let}\ y = s[\vec{r}]\ \mathbf{in}\ e_2 = s'[(\vec{r}, \vec{c})].$$

Define a constant-extension map

$$\phi_{\vec{c}} \colon A_{s;\tau_1} \to A_{s';\tau_2}, \qquad \phi_{\vec{c}}(\vec{r}) := (\vec{r}, \vec{c}).$$

By Lemma F.35, $\phi_{\vec{c}}$ is measurable.
Define the measurable set of hole-assignments corresponding to A at skeleton s':

$$\begin{aligned}
A_{s'} :=& \{(\vec{r}, \vec{r}\,') \in A_{s';\tau_2} \mid s'[\vec{r}, \vec{r}\,'] \in A\} \\
:=& \{(\vec{r}, \vec{r}\,') \in A_{s';\tau_2} \mid s'[\vec{r}, \vec{r}\,'] \in A \cap Expr_{\tau_2}\} && \text{(Definition F.1)} \\
:=& \{(\vec{r}, \vec{r}\,') \in A_{s';\tau_2} \mid s'[\vec{r}, \vec{r}\,'] \in A \cap Expr_{s';\tau_2}\} && (s'[\vec{r}, \vec{r}\,'] \text{ ranges over } Expr_{s';\tau_2})
\end{aligned}$$

Since $A \cap Expr_{s';\tau_2} \in \mathcal{F}_{Expr_{s';\tau_2}}$, by Lemma F.21, $A_{s'} \in \mathcal{F}_{s';\tau_2}$.

Then:

$$\begin{aligned}
R_s &= \{\vec{r} \in A_{s;\tau_1} \mid \mathbf{let}\ y = s[\vec{r}]\ \mathbf{in}\ e_2 \in A\} \\
&= \{\vec{r} \in A_{s;\tau_1} \mid s'[(\vec{r}, \vec{c})] \in A\} \\
&= \{\vec{r} \in A_{s;\tau_1} \mid s'[\phi_{\vec{c}}(\vec{r})] \in A\} \\
&= \{\vec{r} \in A_{s;\tau_1} \mid \phi_{\vec{c}}(\vec{r}) \in A_{s'}\} \\
&= \phi_{\vec{c}}^{-1}(A_{s'}).
\end{aligned}$$

Since $A_{s'} \in \mathcal{F}_{s';\tau_2}$ and $\phi_{\vec{c}}$ is measurable, we conclude

$$\phi_{\vec{c}}^{-1}(A_{s'}) \in \mathcal{F}_{s;\tau_1},$$

and therefore $B \cap Expr_{s;\tau_1} \in \mathcal{F}_{Expr_{s;\tau_1}}$ for all $s \in Skel_{\tau_1}$, giving $B \in \mathcal{F}_{Expr_{\tau_1}}$ as required.

Hence, $\{x \in Expr_{\tau_1} \mid g(x) > a\} \in \mathcal{F}_{Expr_{\tau_1}}$ for all $a \in \mathbb{R}$, so $g^{-1}((a, \infty) \cap [0, 1]) \in \mathcal{F}_{Expr_{\tau_1}}$ for all $a \in \mathbb{R}$. Hence $g^{-1}(B) \in \mathcal{F}_{Expr_{\tau_1}}$ for every $B \in \mathcal{B}([0, 1])$, and g is measurable.

Thus, $f \colon Expr_{\tau_1} \to \mathcal{D}(Expr_{\tau_2} + \bot)$ is a sub-Markov kernel.

- All other cases analogous.

By Lemma F.30, the $\bot$-extension $f_\bot \colon (Expr_{\tau_1} + \bot) \to \mathcal{D}(Expr_{\tau_2} + \bot)$ is also a sub-Markov kernel, as required. □

Lemma F.39 (Small-step semantics is a sub-Markov kernel). *For each type τ, the small-step operational semantics $[\![\cdot]\!] : Expr_\tau \to \mathcal{D}(Expr_\tau + \bot)$ defines a sub-Markov kernel.*

Proof. By induction on the typing derivation of $e : \tau$. We show in each case that the map $e \mapsto [\![e : \tau]\!]$ is a sub-Markov kernel.

- Consider $e = (\mathbf{let}\ y = x : \tau_1\ \mathbf{in}\ e_2 : \tau_2) : \tau_2$. The small-step semantics is:
$$[\![\mathbf{let}\ x = e_1\ \mathbf{in}\ e_2 : \tau_2]\!] = \begin{cases} \delta_{e_2[v/x]} & \text{if } e_1 = v \\ [\![e_1 : \tau_1]\!] \gg\!= \lambda g.\ \delta_{\mathbf{let}\ x=g\ \mathbf{in}\ e_2} & \text{if } e_1 \neq v \end{cases}$$
  - If $e_1 = v$, By Lemma F.31 and Lemma F.37, $e \mapsto \delta_{e_2[v/y]:\tau_2}$ is a sub-Markov kernel. Hence, $[\![\mathbf{let}\ x = e_1\ \mathbf{in}\ e_2 : \tau_2]\!] : Expr_{\tau_2} \to \mathcal{D}(Expr_{\tau_2} + \bot)$ is a sub-Markov kernel.
  - By the induction hypothesis,
$$[\![e_1 : \tau_1]\!] : Expr_{\tau_1} \to \mathcal{D}(Expr_{\tau_1} + \bot)$$
    is a sub-Markov kernel. By Lemma F.38,
$$\lambda g.\ \delta_{\mathbf{let}\ x=g\ \mathbf{in}\ e_2} : (Expr_{\tau_1} + \bot) \to \mathcal{D}(Expr_{\tau_2} + \bot)$$
    is a sub-Markov kernel. Hence by Lemma F.33, their composition
$$[\![e_1 : \tau_1]\!] \gg\!= \lambda g.\ \delta_{\mathbf{let}\ x=g\ \mathbf{in}\ e_2} : Expr_{\tau_1} \to \mathcal{D}(Expr_{\tau_2} + \bot)$$
    is a sub-Markov kernel.
    Thus, $[\![\mathbf{let}\ x = e_1\ \mathbf{in}\ e_2 : \tau_2]\!] : Expr_{\tau_2} \to \mathcal{D}(Expr_{\tau_2} + \bot)$ is a sub-Markov kernel.
- Consider $e = \mathbf{diverge} : \tau$.
  The small-step semantics is:
$$[\![\mathbf{diverge} : \tau]\!] = \lambda A.\ 0.$$
  By Lemma F.32,
$$\lambda A.\ 0$$
  is a sub-Markov kernel.
  Thus, $[\![\mathbf{diverge} : \tau]\!] : Expr_\tau \to \mathcal{D}(Expr_\tau + \bot)$ is a sub-Markov kernel.
- All other cases are analogous.

□

F.5 Small-Step Coupling Semantics

Define $P_\tau = \{(e : \tau, \mathcal{D}[\![e : \tau]\!] : \mathcal{D}[\![\tau]\!]) \mid e : \tau\}$. To define subprobability measures and kernels on the coupling space, we must first equip it with a σ-algebra. Since
$$P_\tau \subseteq Expr_\tau \times Expr_{\mathcal{D}[\![\tau]\!]},$$
we use the subspace σ-algebra from Definition F.1, inherited from the product measurable space. We then extend this space with the isolated error element $\bot$, as for $Expr_\tau + \bot$.

Definition F.40 (σ-algebra construction on P_τ). Let
$$P_\tau = \{(e : \tau, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!]) \mid e : \tau\} \subseteq Expr_\tau \times Expr_{\mathcal{D}[\![\tau]\!]}.$$
We equip P_τ with the subspace σ-algebra
$$\mathcal{F}_{P_\tau} := \left\{ P_\tau \cap A \;\middle|\; A \in \mathcal{F}_{Expr_\tau} \otimes \mathcal{F}_{Expr_{\mathcal{D}[\![\tau]\!]}} \right\}.$$
We equip $P_\tau + \bot$ with
$$\mathcal{F}_{P_\tau + \bot} := \{A,\ A \cup \{\bot\} \mid A \in \mathcal{F}_{P_\tau}\}.$$

Table 3. Small-step coupling semantics. Notice that $\mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!]$ is uniquely determined by $e : \boldsymbol{\tau}$ so that it suffices to indicate the latter in the left-hand column. Here, every function of the form $f(g,g') = \lambda(g,g').\delta_{\ldots}$ is extended to $f_\perp$ in the sense of Definition F.29.

$\mathbf{e} : \boldsymbol{\tau}$	constraints	$[\![e : \tau, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!]]\!]$
$\mathbf{diverge} : \tau$		$\lambda A.0$
$v : \tau$		$\delta_{(v:\tau,\mathcal{D}[\![v:\tau]\!]:\mathcal{D}[\![\tau]\!])}$
$\mathbf{let}\ x = e_1 : \tau_1\ \mathbf{in}\ e_2 : \tau_2 : \tau_2$	$e_1 = v$ e_1 no val.	$\delta_{(e_2[v/x]:\tau_2,\mathcal{D}[\![e_2:\tau_2]\!][\mathcal{D}[\![v:\tau_1]\!]/x]:\mathcal{D}[\![\tau_2]\!])}$ $[\![e_1 : \tau_1, \mathcal{D}[\![e_1 : \tau_1]\!] : \mathcal{D}[\![\tau_1]\!]]\!]$ $\gg\!= \lambda(g,g').\delta_{(\mathbf{let}\ x=g\ \mathbf{in}\ e_2:\tau_2:\tau_2,\mathbf{let}\ x=g'\ \mathbf{in}\ \mathcal{D}[\![e_2:\tau_2]\!]:\mathcal{D}[\![\tau_2]\!]:\mathcal{D}[\![\tau_2]\!])}$
$\mathbf{if}\ e_1 : \mathbf{bool}\ \mathbf{then}\ e_2 : \tau\ \mathbf{else}\ e_3 : \tau : \tau$	$e_1 = \text{true}$ $e_1 = \text{false}$ o.w.	$\delta_{(e_2:\tau,\mathcal{D}[\![e_2:\tau]\!]:\mathcal{D}[\![\tau]\!])}$ $\delta_{(e_3:\tau,\mathcal{D}[\![e_3:\tau]\!]:\mathcal{D}[\![\tau]\!])}$ $[\![e_1 : \mathbf{bool}, \mathcal{D}[\![e_1 : \mathbf{bool}]\!] : \mathcal{D}[\![\mathbf{bool}]\!]]\!]$ $\gg\!= \lambda(g,g').\delta_{(\mathbf{if}\ g\ \mathbf{then}\ e_2:\tau\ \mathbf{else}\ e_3:\tau:\tau,\ \mathbf{if}\ g'\ \mathbf{then}\ \mathcal{D}[\![e_2:\tau]\!]:\mathcal{D}[\![\tau]\!]\ \mathbf{else}\ \mathcal{D}[\![e_3:\tau]\!]:\mathcal{D}[\![\tau]\!]:\mathcal{D}[\![\tau]\!])}$
$\mathbf{observe}\ (e : \mathbf{bool}) : \mathbf{unit}$	$e = \text{true}$ $e = \text{false}$ o.w.	$\delta_{(\mathbf{unit},\mathbf{unit})}$ $\delta_\perp$ $[\![e : \mathbf{bool}, \mathcal{D}[\![e]\!] : \mathbf{bool}]\!]$ $\gg\!= \lambda(g,g').\delta_{(\mathbf{observe}\ (g:\mathbf{bool}):\mathbf{unit},\mathbf{observe}\ (g':\mathbf{bool}):\mathbf{unit})}$
$e_1 : \tau_1 < e_2 : \tau_2 : \mathbf{bool}$ for $\tau_1 = \mathbf{float}[B;V_1], \tau_2 = \mathbf{float}[B;V_2]$ $B = \top$	$e_1 = v_1, e_2 = v_2, v_1 < v_2$ $e_1 = v_1, e_2 = v_2, v_1 \geq v_2$ $e_1 = v_1, e_2 \neq v_2$ o.w.	$\delta_{(\text{true}:\mathbf{bool},\text{true}:\mathbf{bool})}$ $\delta_{(\text{false}:\mathbf{bool},\text{false}:\mathbf{bool})}$ $[\![e_2 : \tau_2, \mathcal{D}[\![e_2 : \tau_2]\!] : \mathcal{D}[\![\tau_2]\!]]\!]$ $\gg\!= \lambda(g,g').\delta_{(v_1:\tau_1<g:\mathbf{bool},\mathcal{D}[\![v_1:\tau_1]\!]:\mathcal{D}[\![\tau_1]\!]<g':\mathcal{D}[\![\mathbf{bool}]\!])}$ $[\![e_1 : \tau_1, \mathcal{D}[\![e_1 : \tau_1]\!] : \mathcal{D}[\![\tau_1]\!]]\!]$ $\gg\!= \lambda(g,g').\delta_{(g<e_2:\tau_2:\mathbf{bool},g'<\mathcal{D}[\![e_2:\tau_2]\!]:\mathcal{D}[\![\tau_2]\!]:\mathcal{D}[\![\mathbf{bool}]\!])}$
$e_1 : \tau_1 < e_2 : \tau_2 : \mathbf{bool}$ for $\tau_1 = \mathbf{float}[B;V_1], \tau_2 = \mathbf{float}[B;V_2]$ $B \neq \top$	$e_1 = v_1, e_2 = v_2, v_1 < v_2$ $e_1 = v_1, e_2 = v_2, v_1 \geq v_2$ $e_1 = v_1, e_2 \neq v_2$ o.w.	$\delta_{(\text{true}:\mathbf{bool},\text{true}:\mathbf{bool})}$ $\delta_{(\text{false}:\mathbf{bool},\text{false}:\mathbf{bool})}$ $[\![e_2 : \tau_2, \mathcal{D}[\![e_2 : \tau_2]\!] : \mathcal{D}[\![\tau_2]\!]]\!]$ $\gg\!= \lambda(g,g').\delta_{(v_1:\tau_1<g:\mathbf{bool},\mathcal{D}[\![v_1:\tau_1]\!]:\mathcal{D}[\![\tau_1]\!]<_{\#(\lvert B\rvert+1)}g':\mathcal{D}[\![\mathbf{bool}]\!])}$ $[\![e_1 : \tau_1, \mathcal{D}[\![e_1 : \tau_1]\!] : \mathcal{D}[\![\tau_1]\!]]\!]$ $\gg\!= \lambda(g,g').\delta_{(g<e_2:\tau_2:\mathbf{bool},g'<_{\#(\lvert B\rvert+1)}\mathcal{D}[\![e_2:\tau_2]\!]:\mathcal{D}[\![\tau_2]\!]:\mathcal{D}[\![\mathbf{bool}]\!])}$
$\mathbf{uniform}(e_1 : \tau_1, e_2 : \tau_2) : \mathbf{float}[B;V]$ $B = \top$	$e_1 = v_1 < v_2 = e_2$ $e_1 = v_1 \geq v_2 = e_2$ $e_1 = v_1$, e_2 no val. e_1 no val.	$\text{Uniform}(v_1, v_2) \gg\!= \lambda v.\delta_{(v:\mathbf{float}[B;V],v:\mathbf{float}[B;V])}$ $[\![\mathbf{diverge} : \tau, \mathcal{D}[\![\mathbf{uniform}(v_1 : \tau_1, v_2 : \tau_2)]\!] : \mathcal{D}[\![\tau]\!]]\!]$ $[\![e_2 : \tau_2, \mathcal{D}[\![e_2]\!] : \mathcal{D}[\![\tau_2]\!]]\!]$ $\gg\!= \lambda(g,g').\delta_{(\mathbf{uniform}(v_1:\tau_1,g:\tau_2):\tau,\mathbf{uniform}(\mathcal{D}[\![v_1]\!]:\mathcal{D}[\![\tau_1]\!],g':\mathcal{D}[\![\tau_2]\!]):\mathcal{D}[\![\tau]\!])}$ $[\![e_1 : \tau_1, \mathcal{D}[\![e_1]\!] : \mathcal{D}[\![\tau_1]\!]]\!]$ $\gg\!= \lambda(g,g').\delta_{(\mathbf{uniform}(g:\tau_1,e_2:\tau_2):\tau,\mathbf{uniform}(g':\mathcal{D}[\![\tau_1]\!],\mathcal{D}[\![e_2]\!]:\mathcal{D}[\![\tau_2]\!]):\mathcal{D}[\![\tau]\!])}$
$\mathbf{uniform}(e_1 : \tau_1, e_2 : \tau_2) : \mathbf{float}[B;V]$ $B \neq \top$ $\tau_1 = \mathbf{float}[B_1 \neq \top, \{v_1, \ldots, v_n\}]$ $\tau_2 = \mathbf{float}[B_2 \neq \top, \{w_1, \ldots, w_m\}]$	$e_1 = c_1 < c_2 = e_2$ values $e_1 = v_1 \geq v_2 = e_2$ $e_1 = c_1$, e_2 no val. e_1 no val.	$\text{Uniform}(c_1, c_2) \gg\!= \lambda v.\delta_{(v:\mathbf{float}[B;V],v_i:\mathbf{fin}(\lvert B\rvert+1))}$ where v_i is unique number s.t. $v \in \text{Intervals}(B)_{v_i}$ $[\![\mathbf{diverge} : \tau, \mathcal{D}[\![\mathbf{uniform}(v_1 : \tau_1, v_2 : \tau_2)]\!] : \mathcal{D}[\![\tau]\!]]\!]$ $[\![e_2 : \tau_2, \mathcal{D}[\![e_2]\!] : \mathcal{D}[\![\tau_2]\!]]\!]$ $\gg\!= \lambda(g,g').\delta_{(\mathbf{uniform}(v_1:\tau_1,g:\tau_2):\tau,\mathtt{DC}_{B,B_1,B_2,V_1,V_2}(\mathcal{D}[\![v_1]\!]:\mathcal{D}[\![\tau_1]\!],g':\mathcal{D}[\![\tau_2]\!]):\mathcal{D}[\![\tau]\!])}$ $[\![e_1 : \tau_1, \mathcal{D}[\![e_1]\!] : \mathcal{D}[\![\tau_1]\!]]\!]$ $\gg\!= \lambda(g,g').\delta_{(\mathbf{uniform}(g:\tau_1,e_2:\tau_2):\tau,\mathtt{DC}_{B,B_1,B_2,V_1,V_2}(g':\mathcal{D}[\![\tau_1]\!],\mathcal{D}[\![e_2]\!]:\mathcal{D}[\![\tau_2]\!]):\mathcal{D}[\![\tau]\!])}$

Lemma F.41. *$(P_\tau, \mathcal{F}_{P_\tau})$ and $(P_\tau + \perp, \mathcal{F}_{P_\tau+\perp})$ are measurable spaces.*

Proof. The first claim follows from Definition F.1. The second follows from the same error-extension construction used for $Expr_\tau + \perp$. □

Now, exploiting that $e : \tau$ is a value iff $\mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!]$ is a value, we define:

Definition F.42. The small-step coupling $[\![e : \tau, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!]]\!] \in \mathcal{D}(P_\tau + \perp)$ is defined recursively on the structure of $e : \tau$ by the rules in Tables 3 and 4.

Lemma F.43 (Monadic bind is a sub-Markov kernel). *For every occurrence of a monadic bind $\mu \gg\!= (f, f')$ given in the small-step operational coupling semantics in Tables 3 and 4, $(f, f') : (P_{\tau_1} + \perp) \to \mathcal{D}(P_{\tau_2} + \perp)$ is a sub-Markov kernel.*

Table 4. Small-step coupling semantics (continued). Here, every function of the form $f(g,g') = \lambda(g,g').\delta_{\dots}$ is extended to $f_\bot$ in the sense of Definition F.29.

$\mathbf{e} : \boldsymbol{\tau}$	constraints	$[\![\boldsymbol{e} : \boldsymbol{\tau}, \mathcal{D}[\![\boldsymbol{e}]\!] : \mathcal{D}[\![\boldsymbol{\tau}]\!]]\!]$
$e_1 : \tau_1$ && $e_2 : \tau_2$: **bool** $\tau_1 =$ **bool**, $\tau_2 =$ **bool**	$e_1 = b_1 =$ true, $e_2 = b_2 =$ true $e_1 = b_1$, $e_2 = b_2$ $e_1 = b_1$, $e_2 \neq b_2$ o.w.	$\delta_{(\text{true}:\mathbf{bool},\text{true}:\mathbf{bool})}$ $\delta_{(\text{false}:\mathbf{bool},\text{false}:\mathbf{bool})}$ $[\![e_2 : \tau_2, \mathcal{D}[\![e_2 : \tau_2]\!] : \mathcal{D}[\![\tau_2]\!]]\!]$ $\gg\!= \lambda(g,g').\delta_{(b_1:\tau_1 \&\& g:\mathbf{bool},\ b_1:\mathcal{D}[\![\tau_1]\!] \&\& g':\mathcal{D}[\![\mathbf{bool}]\!])}$ $[\![e_1 : \tau_1, \mathcal{D}[\![e_1 : \tau_1]\!] : \mathcal{D}[\![\tau_1]\!]]\!]$ $\gg\!= \lambda(g,g').\delta_{(g \&\& e_2:\tau_2:\mathbf{bool},\ g' \&\& \mathcal{D}[\![e_2:\tau_2]\!]:\mathcal{D}[\![\tau_2]\!]:\mathcal{D}[\![\mathbf{bool}]\!])}$
$e_1 : \tau_1$ || $e_2 : \tau_2$: **bool** $\tau_1 =$ **bool**, $\tau_2 =$ **bool**	$e_1 = b_1 =$ false, $e_2 = b_2 =$ false $e_1 = b_1$, $e_2 = b_2$ $e_1 = b_1$, $e_2 \neq b_2$ o.w.	$\delta_{(\text{false}:\mathbf{bool},\text{false}:\mathbf{bool})}$ $\delta_{(\text{true}:\mathbf{bool},\text{true}:\mathbf{bool})}$ $[\![e_2 : \tau_2, \mathcal{D}[\![e_2 : \tau_2]\!] : \mathcal{D}[\![\tau_2]\!]]\!]$ $\gg\!= \lambda(g,g').\delta_{(b_1:\tau_1 \,\|\|\, g:\mathbf{bool},\ b_1:\mathcal{D}[\![\tau_1]\!] \,\|\|\, g':\mathcal{D}[\![\mathbf{bool}]\!])}$ $[\![e_1 : \tau_1, \mathcal{D}[\![e_1 : \tau_1]\!] : \mathcal{D}[\![\tau_1]\!]]\!]$ $\gg\!= \lambda(g,g').\delta_{(g \,\|\|\, e_2:\tau_2:\mathbf{bool},\ g' \,\|\|\, \mathcal{D}[\![e_2:\tau_2]\!]:\mathcal{D}[\![\tau_2]\!]:\mathcal{D}[\![\mathbf{bool}]\!])}$
not $e : \tau$: **bool** $\tau =$ **bool**	$e =$ true $e =$ false o.w.	$\delta_{(\text{false}:\mathbf{bool},\text{false}:\mathbf{bool})}$ $\delta_{(\text{true}:\mathbf{bool},\text{true}:\mathbf{bool})}$ $[\![e : \tau, \mathcal{D}[\![e : \tau]\!] : \mathcal{D}[\![\tau]\!]]\!]$ $\gg\!= \lambda(g,g').\delta_{(\text{not } g:\mathbf{bool},\ \text{not } g':\mathcal{D}[\![\mathbf{bool}]\!])}$
$(e_1 : \tau_1, e_2 : \tau_2) : \tau_1 * \tau_2$	$e_1 = v_1$, $e_2 = v_2$ $e_1 = v_1$, $e_2 \neq v_2$ o.w.	$\delta_{((v_1,v_2):\tau_1*\tau_2,\ (\mathcal{D}[\![v_1]\!]:\mathcal{D}[\![\tau_1]\!], \mathcal{D}[\![v_2]\!]:\mathcal{D}[\![\tau_2]\!]):\mathcal{D}[\![\tau_1]\!]*\mathcal{D}[\![\tau_2]\!])}$ $[\![e_2 : \tau_2, \mathcal{D}[\![e_2 : \tau_2]\!] : \mathcal{D}[\![\tau_2]\!]]\!]$ $\gg\!= \lambda(g,g').\delta_{((v_1:\tau_1,g:\tau_2):\tau_1*\tau_2,\ (\mathcal{D}[\![v_1]\!]:\mathcal{D}[\![\tau_1]\!], g':\mathcal{D}[\![\tau_2]\!]):\mathcal{D}[\![\tau_1]\!]*\mathcal{D}[\![\tau_2]\!])}$ $[\![e_1 : \tau_1, \mathcal{D}[\![e_1 : \tau_1]\!] : \mathcal{D}[\![\tau_1]\!]]\!]$ $\gg\!= \lambda(g,g').\delta_{((g,e_2:\tau_2):\tau_1*\tau_2,\ (g', \mathcal{D}[\![e_2:\tau_2]\!]):\mathcal{D}[\![\tau_1]\!]*\mathcal{D}[\![\tau_2]\!])}$
fst $(e : \tau_1 * \tau_2) : \tau_1$	$e = (v_1, v_2)$ o.w.	$\delta_{(v_1:\tau_1,\ \mathcal{D}[\![v_1:\tau_1]\!]:\mathcal{D}[\![\tau_1]\!])}$ $[\![e : \tau_1 * \tau_2, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau_1 * \tau_2]\!]]\!]$ $\gg\!= \lambda(g,g').\delta_{(\mathbf{fst}\ (g:\tau_1*\tau_2):\tau_1,\ \mathbf{fst}\ (g':\mathcal{D}[\![\tau_1*\tau_2]\!]):\mathcal{D}[\![\tau_1]\!])}$
snd $(e : \tau_1 * \tau_2) : \tau_2$	$e = (v_1, v_2)$ o.w.	$\delta_{(v_2:\tau_2,\ \mathcal{D}[\![v_2:\tau_2]\!]:\mathcal{D}[\![\tau_2]\!])}$ $[\![e : \tau_1 * \tau_2, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau_1 * \tau_2]\!]]\!]$ $\gg\!= \lambda(g,g').\delta_{(\mathbf{snd}\ (g:\tau_1*\tau_2):\tau_2,\ \mathbf{snd}\ (g':\mathcal{D}[\![\tau_1*\tau_2]\!]):\mathcal{D}[\![\tau_2]\!])}$
fun $x \to e : \tau_1 \to \tau_2$		$\delta_{(\mathbf{fun}\ x \to e:\tau_1\to\tau_2,\ \mathbf{fun}\ x \to \mathcal{D}[\![e]\!]:\mathcal{D}[\![\tau_1]\!]\to\mathcal{D}[\![\tau_2]\!])}$
fix $g\ x := e : \tau_1 \to \tau_2$		$\delta_{(\mathbf{fix}\ g\ x:=e:\tau_1\to\tau_2,\ \mathbf{fix}\ g\ x:=\mathcal{D}[\![e]\!]:\mathcal{D}[\![\tau_1]\!]\to\mathcal{D}[\![\tau_2]\!])}$
$(e_1 : \tau_1 \to \tau_2 \ \ e_2 : \tau_1) : \tau_2$	$e_1 = \mathbf{fun}\ x \to e$ or $\mathbf{fix}\ f\ x := e$, $e_2 = v_2$ $e_1 = \mathbf{fun}\ x \to e$ or $\mathbf{fix}\ f\ x := e$, $e_2 \neq v_2$ o.w.	$\delta_{(e[v_2/x]:\tau_2,\ \mathcal{D}[\![e]\!][\mathcal{D}[\![v_2]\!]/x]:\mathcal{D}[\![\tau_2]\!])}$ $[\![e_2 : \tau_1, \mathcal{D}[\![e_2 : \tau_1]\!] : \mathcal{D}[\![\tau_1]\!]]\!]$ $\gg\!= \lambda(g,g').\delta_{(e_1:\tau_1\to\tau_2\ \ g:\tau_1:\tau_2,\ \mathcal{D}[\![e_1]\!]:\mathcal{D}[\![\tau_1]\!]\to\mathcal{D}[\![\tau_2]\!]\ \ g':\mathcal{D}[\![\tau_1]\!]:\mathcal{D}[\![\tau_2]\!])}$ $[\![e_1 : \tau_1 \to \tau_2, \mathcal{D}[\![e_1]\!] : \mathcal{D}[\![\tau_1 \to \tau_2]\!]]\!]$ $\gg\!= \lambda(g,g').\delta_{(g\ e_2:\tau_1:\tau_2,\ g'\ \mathcal{D}[\![e_2]\!]:\mathcal{D}[\![\tau_1]\!]:\mathcal{D}[\![\tau_2]\!])}$
nil : $\mathbf{list}(\tau)$		$\delta_{(\text{nil}:\mathbf{list}(\tau),\ \text{nil}:\mathbf{list}(\mathcal{D}[\![\tau]\!]))}$
$(e_1 : \tau :: e_2 : \mathbf{list}(\tau)) : \mathbf{list}(\tau)$	$e_1 = v_1$, $e_2 = v_2$ $e_1 = v_1$, $e_2 \neq v_2$ o.w.	$\delta_{(v_1::v_2:\mathbf{list}(\tau),\ \mathcal{D}[\![v_1]\!]::\mathcal{D}[\![v_2]\!]:\mathbf{list}(\mathcal{D}[\![\tau]\!]))}$ $[\![e_2 : \mathbf{list}(\tau), \mathcal{D}[\![e_2]\!] : \mathbf{list}(\mathcal{D}[\![\tau]\!])]\!]$ $\gg\!= \lambda(g,g').\delta_{(v_1::g:\mathbf{list}(\tau),\ \mathcal{D}[\![v_1]\!]::g':\mathbf{list}(\mathcal{D}[\![\tau]\!]))}$ $[\![e_1 : \tau, \mathcal{D}[\![e_1]\!] : \mathcal{D}[\![\tau]\!]]\!]$ $\gg\!= \lambda(g,g').\delta_{(g::e_2:\mathbf{list}(\tau):\mathbf{list}(\tau),\ g'::\mathcal{D}[\![e_2]\!]:\mathbf{list}(\mathcal{D}[\![\tau]\!]):\mathbf{list}(\mathcal{D}[\![\tau]\!]))}$
(**match** $e : \mathbf{list}(\tau_1)$ **with** | nil $\to e_1 : \tau_2$ | $h :: t \to e_2 : \tau_2$ **end**) : τ_2	$e =$ nil $e = v_h :: v_t$ o.w.	$\delta_{(e_1:\tau_2,\ \mathcal{D}[\![e_1:\tau_2]\!]:\mathcal{D}[\![\tau_2]\!])}$ $\delta_{(e_2[v_h/h,v_t/t]:\tau_2,\ \mathcal{D}[\![e_2:\tau_2]\!][\mathcal{D}[\![v_h]\!]/h,\mathcal{D}[\![v_t]\!]/t]:\mathcal{D}[\![\tau_2]\!])}$ $[\![e : \mathbf{list}(\tau_1), \mathcal{D}[\![e]\!] : \mathbf{list}(\mathcal{D}[\![\tau_1]\!])]\!]$ $\gg\!= \lambda(g,g').\delta_{(\mathbf{match}\ g\ \mathbf{with}\ \|\text{nil}\to e_1:\tau_2\ \|h::t\to e_2:\tau_2\ \mathbf{end}:\tau_2,\ \mathbf{match}\ g'\ \mathbf{with}\ \|\text{nil}\to\mathcal{D}[\![e_1:\tau_2]\!]:\mathcal{D}[\![\tau_2]\!]\ \|h::t\to\mathcal{D}[\![e_2:\tau_2]\!]:\mathcal{D}[\![\tau_2]\!]\ \mathbf{end}:\mathcal{D}[\![\tau_2]\!])}$

Proof. Analogous to proof of Lemma F.38. □

Lemma F.44 (Small-step coupling semantics is a sub-Markov kernel). *For each type τ, the small-step operational coupling semantics $[\![\cdot,\cdot]\!] : P_\tau \to \mathcal{D}(P_\tau + \bot)$ defines a sub-Markov kernel.*

Proof. Analogous to proof of Lemma F.39. □

We now define suitable projection functions on measures $\mu \in \mathcal{D}(P_\tau + \bot)$ by

$$\pi_1(\mu) = \mu \gg\!= \lambda(e_1, e_2).\delta_{e_1} \tag{1}$$
$$\pi_2(\mu) = \mu \gg\!= \lambda(e_1, e_2).\delta_{e_2}\ . \tag{2}$$

We have the following key property:

Lemma F.45 (Small-step coupling projection). *Let $e : \tau$. We have:*

(1) $\pi_1([\![e : \tau, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!]]\!]) = [\![e : \tau]\!]$.
(2) $\pi_2([\![e : \tau, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!]]\!]) = [\![\mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!]]\!]$.

Proof. By induction on the type derivation of $e : \tau$. The most interesting case is

$$e : \tau = \mathbf{uniform}(c_1 : \mathbf{float}[B_1; V_1], c_2 : \mathbf{float}[B_2; V_2]) : \mathbf{float}[B; V]\ ,$$

where

(1) $c_1 < c_2$,
(2) $B = \{\sim_1 b_1, \ldots, \sim_n b_n\}$,
(3) $V = \top$,
(4) $V_1 = \{v_1, \ldots, v_{n_1}\}$ with $\mathrm{recovers}(B_1, V_1)$ and $c_1 \in V_1$ due to $\mathrm{has}(V_1, c_1)$ by I.H.
(5) $V_2 = \{w_1, \ldots, w_{n_2}\}$ with $\mathrm{recovers}(B_2, V_2)$ and $c_2 \in V_2$ due to $\mathrm{has}(V_2, c_2)$ by I.H.

First observe that we have

$$\begin{aligned}
&\mathcal{D}[\![e : \tau]\!] : \mathcal{D}[\![\tau]\!] \\
&= \mathtt{DC}_{B,B_1,B_2,V_1,V_2}(\mathcal{D}[\![c_1 : \mathbf{float}[B_1; V_1]]\!], \mathcal{D}[\![c_2 : \mathbf{float}[B_2; V_2]]\!]) : \mathbf{fin}(|B| + 1) \\
&= \mathtt{DC}_{B,B_1,B_2,V_1,V_2}(k_1 : \mathbf{fin}(|B_1| + 1), k_2 : \mathbf{fin}(|B_2| + 1)) : \mathbf{fin}(|B| + 1)\ ,
\end{aligned}$$

where k_i is the unique number such that $c_i \in \mathrm{Intervals}(B_i)_{k_i}$. By Table 1 and Figure 13, this yields

$$\begin{aligned}
&[\![\mathcal{D}[\![e : \tau]\!]]\!] \\
&= [\![\mathbf{discrete}(p_0, \ldots, p_n)]\!] \text{ where } p_k = \mathrm{CDF}(b_{k+1}) - \mathrm{CDF}(b_k) = \frac{b_{k+1} - c_1}{c_2 - c_1} - \frac{b_k - c_1}{c_2 - c_1} = \frac{b_{k+1} - b_k}{c_2 - c_1}\ .
\end{aligned}$$

For Lemma F.45.2, consider the following:

$$\begin{aligned}
&\pi_1([\![e : \tau, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!]]\!]) \\
&= [\![e : \tau, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!]]\!] \gg\!= \lambda(e_1, e_2).\delta_{e_1} && \text{(definition)} \\
&= (\mathrm{Uniform}(c_1, c_2) \gg\!= \lambda v.\delta_{(v:\mathbf{float}[B;V], v_i:\mathbf{fin}(|B|+1))}) \gg\!= \lambda(e_1, e_2).\delta_{e_1} \\
&\qquad\qquad \text{(Table 3, } v_i \text{ is unique number s.t. } v \in \mathrm{Intervals}(B)_{v_i}) \\
&= \mathrm{Uniform}(c_1, c_2) \gg\!= \lambda v.(\delta_{(v:\mathbf{float}[B;V], v_i:\mathbf{fin}(|B|+1))} \gg\!= \lambda(e_1, e_2).\delta_{e_1}) \\
&\qquad\qquad \text{(monad assoc., } v_i \text{ is unique number s.t. } v \in \mathrm{Intervals}(B)_{v_i}) \\
&= \mathrm{Uniform}(c_1, c_2) \gg\!= \lambda v.\delta_{v:\mathbf{float}[B;V]} \\
&= \mathrm{Uniform}(c_1, c_2) \\
&= [\![e : \tau]\!]\ . && \text{(Table 1)}
\end{aligned}$$

For Lemma F.45.2, consider the following:

$$
\begin{aligned}
& \pi_2(\llbracket e : \tau, \mathcal{D}\llbracket e \rrbracket : \mathcal{D}\llbracket \tau \rrbracket \rrbracket) \\
&= \llbracket e : \tau, \mathcal{D}\llbracket e \rrbracket : \mathcal{D}\llbracket \tau \rrbracket \rrbracket \gg\!= \lambda(e_1, e_2).\delta_{e_2} && \text{(definition)} \\
&= (\text{Uniform}(c_1, c_2) \gg\!= \lambda v.\delta_{(v:\mathbf{float}[B;V],v_i:\mathbf{fin}(|B|+1))}) \gg\!= \lambda(e_1, e_2).\delta_{e_2} \\
& && \text{(Table 3, } v_i \text{ is unique number s.t. } v \in \text{Intervals}(B)_{v_i}) \\
&= \text{Uniform}(c_1, c_2) \gg\!= \lambda v.(\delta_{(v:\mathbf{float}[B;V],v_i:\mathbf{fin}(|B|+1))} \gg\!= \lambda(e_1, e_2).\delta_{e_2}) \\
& && \text{(monad assoc., } v_i \text{ is unique number s.t. } v \in \text{Intervals}(B)_{v_i}) \\
&= \text{Uniform}(c_1, c_2) \gg\!= \lambda v.\delta_{v_i:\mathbf{fin}(|B|+1)} \\
& && \text{(monad return, } v_i \text{ is unique number s.t. } v \in \text{Intervals}(B)_{v_i}) \\
&= \llbracket \mathbf{discrete}(p_0, \ldots, p_n) \rrbracket \text{ where } p_k = \text{CDF}(b_{k+1}) - \text{CDF}(b_k) = \frac{b_{k+1} - c_1}{c_2 - c_1} - \frac{b_k - c_1}{c_2 - c_1} = \frac{b_{k+1} - b_k}{c_2 - c_1} \\
& && \text{(by construction)} \\
&= \llbracket \mathcal{D}\llbracket e \rrbracket : \mathcal{D}\llbracket \tau \rrbracket \rrbracket\,. && \text{(Table 1)}
\end{aligned}
$$

Now, the remaining cases are either (i) trivial base cases or (ii) immediately follow from the I.H. and Lemma F.46 further below. We demonstrate this for the case

$$e : \tau = \mathbf{uniform}(e_1 : \mathbf{float}[B_1; V_1], e_2 : \mathbf{float}[B_2; V_2]) : \mathbf{float}[B; V]\,,$$

where

(1) $e_1 = c_1$ is a value but e_2 is no value,
(2) $B = \{\sim_1 b_1, \ldots, \sim_n b_n\}$,
(3) $V = \top$,
(4) $V_1 = \{v_1, \ldots, v_{n_1}\}$ with recovers(B_1, V_1) and $c_1 \in V_1$ due to has(V_1, c_1) by I.H.
(5) $V_2 = \{w_1, \ldots, w_{n_2}\}$ with recovers(B_2, V_2) and $c_2 \in V_2$ due to has(V_2, c_2) by I.H.

For Lemma F.45.2, consider the following:

$$
\begin{aligned}
& \pi_1(\llbracket e : \tau, \mathcal{D}\llbracket e \rrbracket : \mathcal{D}\llbracket \tau \rrbracket \rrbracket) \\
&= \llbracket e : \tau, \mathcal{D}\llbracket e \rrbracket : \mathcal{D}\llbracket \tau \rrbracket \rrbracket \gg\!= \lambda(e_1, e_2).\delta_{e_1} && \text{(definition)} \\
&= \big(\llbracket e_2 : \tau_2, \mathcal{D}\llbracket e_2 \rrbracket : \mathcal{D}\llbracket \tau_2 \rrbracket \rrbracket \\
& \quad \gg\!= \lambda(f, f').\delta_{(\mathbf{uniform}(v_1:\tau_1, f:\tau_2):\tau, \mathsf{DC}_{B,B_1,B_2,V_1,V_2}(\mathcal{D}\llbracket v_1 \rrbracket : \mathcal{D}\llbracket \tau_1 \rrbracket, f' : \mathcal{D}\llbracket \tau_2 \rrbracket) : \mathcal{D}\llbracket \tau \rrbracket)}\big) \gg\!= \lambda(e_1, e_2).\delta_{e_1} \\
&= \llbracket e_2 : \tau_2, \mathcal{D}\llbracket e_2 \rrbracket : \mathcal{D}\llbracket \tau_2 \rrbracket \rrbracket \\
& \quad \gg\!= \lambda(f, f').\big(\delta_{(\mathbf{uniform}(v_1:\tau_1, f:\tau_2):\tau, \mathsf{DC}_{B,B_1,B_2,V_1,V_2}(\mathcal{D}\llbracket v_1 \rrbracket : \mathcal{D}\llbracket \tau_1 \rrbracket, f' : \mathcal{D}\llbracket \tau_2 \rrbracket) : \mathcal{D}\llbracket \tau \rrbracket)} \gg\!= \lambda(e_1, e_2).\delta_{e_1}\big) \\
& && \text{(monad assoc.)} \\
&= \llbracket e_2 : \tau_2, \mathcal{D}\llbracket e_2 \rrbracket : \mathcal{D}\llbracket \tau_2 \rrbracket \rrbracket \gg\!= \lambda(f, f').\delta_{\mathbf{uniform}(v_1:\tau_1, f:\tau_2):\tau} && \text{(monad return)} \\
&= \pi_1(\llbracket e_2 : \tau_2, \mathcal{D}\llbracket e_2 \rrbracket : \mathcal{D}\llbracket \tau_2 \rrbracket \rrbracket) \gg\!= \lambda f.\delta_{\mathbf{uniform}(v_1:\tau_1, f:\tau_2):\tau} && \text{(Lemma F.46.1)} \\
&= \llbracket e_2 : \tau_2 \rrbracket \gg\!= \lambda f.\delta_{\mathbf{uniform}(v_1:\tau_1, f:\tau_2):\tau}\,. && (\llbracket e_2 : \tau_2, \mathcal{D}\llbracket e_2 \rrbracket : \mathcal{D}\llbracket \tau_2 \rrbracket \rrbracket = \llbracket e_2 : \tau_2 \rrbracket \text{ by I.H.}) \\
&= \llbracket e : \tau \rrbracket && \text{(Table 1)}
\end{aligned}
$$

□

Lemma F.46. *Let* $F\colon P_\tau \to \mathcal{D}(P_\tau + \bot)$, *let* $G\colon Expr_\tau \to \mathcal{D}(Expr_\tau + \bot)$, *and let* $G'\colon Expr_{\mathcal{D}\llbracket\tau\rrbracket} \to \mathcal{D}(Expr_{\mathcal{D}\llbracket\tau\rrbracket} + \bot)$. *We have:*

(1) If $\forall (f, f') \in P_\tau\colon F(f, f') = G(f)$, *then for all* $\mu \in \mathcal{D}(P_\tau + \bot)$,

$$\pi_1(\mu) \gg\!= \lambda f.G(f) \;=\; \mu \gg\!= \lambda(f, f').F(f, f')\,.$$

(2) If $\forall (f, f') \in P_\tau \colon F(f,f') = G'(f)$*, then for all* $\mu \in \mathcal{D}(P_\tau + \bot)$*,*

$$\pi_2(\mu) \gg\!= \lambda f.G'(f) \;=\; \mu \gg\!= \lambda(f,f').F(f,f') \;.$$

Proof. We only prove Lemma F.46.1 because the reasoning for Lemma F.46.2 is completely analogous. We have

$$\begin{aligned}
& \pi_1(\mu) \gg\!= \lambda f.G(f) \\
=\;& (\mu \gg\!= \lambda(f,f').\delta_f) \gg\!= \lambda f.G(f) && \text{(definition)} \\
=\;& \mu \gg\!= \lambda(f,f').(\delta_f \gg\!= \lambda f.G(f)) && \text{(monad assoc.)} \\
=\;& \mu \gg\!= \lambda(f,f').G(f) && \text{(monad neutral)} \\
=\;& \mu \gg\!= \lambda(f,f').F(f,f') \;. && \text{(assumption)}
\end{aligned}$$

□

F.6 Big-Step Semantics and Correctness of Discretization

We extend both of the small-step semantics canonically to an n-step semantics by

$$\langle e : \tau \rangle_n = \begin{cases} \delta_{e:\tau} & \text{if } n = 0 \\ \langle e : \tau \rangle_{n-1} \gg\!= \lambda e'.[\![e' : \tau]\!] & \text{if } n > 0 \;. \end{cases}$$

$$\langle e : \tau, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!] \rangle_n = \begin{cases} \delta_{(e:\tau, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!])} & \text{if } n = 0 \\ \langle e : \tau, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!] \rangle_{n-1} \gg\!= \lambda(e', e'').[\![e' : \tau, e'' : \mathcal{D}[\![\tau]\!]]\!] & \text{if } n > 0 \;. \end{cases}$$

As a sanity check, we show that the n-step semantics defines a sub-Markov kernel.

Lemma F.47 (n-step semantics is a sub-Markov kernel). *For all* $n \in \mathbb{N}$*,* $\langle \cdot \rangle_n : Expr_\tau \to \mathcal{D}(Expr_\tau + \bot)$ *is a sub-Markov kernel.*

Proof. By induction on n.

Base case: $n = 0$. By definition,

$$\langle e : \tau \rangle_n = \delta_{e:\tau}$$

i.e. the map $e \mapsto \delta_e$. By Lemma F.31, this is a sub-Markov kernel.

Inductive step: $n > 0$. By definition,

$$\langle e : \tau \rangle_n = \langle e : \tau \rangle_{n-1} \gg\!= \lambda e'.[\![e' : \tau]\!]$$

By the I.H., $\langle \cdot \rangle_{n-1} \colon Expr_\tau \to \mathcal{D}(Expr_\tau + \bot)$ is a sub-Markov kernel. By Lemma F.39, $\lambda e'.[\![e' : \tau]\!] : Expr_\tau \to \mathcal{D}(Expr_\tau + \bot)$ is a sub-Markov kernel. By Lemma F.30, its $\bot$-extension from $(Expr_\tau + \bot)$ to $\mathcal{D}(Expr_\tau + \bot)$ is also a sub-Markov kernel. By Lemma F.33, $\langle e : \tau \rangle_{n-1} \gg\!= \lambda e'.[\![e' : \tau]\!]$ is a sub-Markov kernel. □

Lemma F.48 (n-step coupling semantics is a sub-Markov kernel). *For all* $n \in \mathbb{N}$*,* $\langle \cdot, \cdot \rangle_n : P_\tau \to \mathcal{D}(P_\tau + \bot)$ *is a sub-Markov kernel.*

Proof. Analogous to proof of Lemma F.47. □

For every type τ, we denote by $Vals_\tau$ the set of values of type τ. By Lemma F.27 and closure under finite unions, $Vals_\tau + \bot$ and $Expr_\tau \setminus Vals_\tau$ are measurable subsets of $Expr_\tau + \bot$, which we equip with their subspace σ-algebras.

Furthermore, let

$$K_\tau : (Expr_\tau + \bot) \to \mathcal{D}(Expr_\tau + \bot)$$

be the $\bot$-extension of the small-step kernel:

$$K_\tau(x, A) = \begin{cases} [\![x : \tau]\!](A) & x \in Expr_\tau, \\ \delta_\bot(A) & x = \bot, \end{cases} \qquad A \in \mathcal{F}_{Expr_\tau + \bot}.$$

Thus, for every $e : \tau$,

$$\langle e : \tau \rangle_0 = \delta_e$$

and

$$\langle e : \tau \rangle_{n+1}(A) = \int_{Expr_\tau + \bot} K_\tau(x, A)\, \langle e : \tau \rangle_n(dx).$$

Lemma F.49 (Monotonicity of n-step semantics). *For every $e : \tau$, $n \in \mathbb{N}$, and $A \in \mathcal{F}_{Vals_\tau + \bot}$,*

$$\langle e : \tau \rangle_n(A) \leq \langle e : \tau \rangle_{n+1}(A).$$

Proof. Fix $e : \tau$, $n \in \mathbb{N}$, and $A \in \mathcal{F}_{Vals_\tau + \bot}$. Write

$$\mu_n := \langle e : \tau \rangle_n.$$

Every $x \in Vals_\tau + \bot$ is absorbing:

$$K_\tau(x, -) = \delta_x.$$

Indeed, $[\![v : \tau]\!] = \delta_v$ for every $v \in Vals_\tau$, while the $\bot$-extension maps $\bot$ to $\delta_\bot$. Therefore,

$$\begin{aligned}
\langle e : \tau \rangle_{n+1}(A) &= \int_{Expr_\tau + \bot} K_\tau(x, A)\, \mu_n(dx) && \text{(definition of the } n\text{-step semantics)} \\
&= \int_{Vals_\tau + \bot} K_\tau(x, A)\, \mu_n(dx) + \int_{Expr_\tau \setminus Vals_\tau} K_\tau(x, A)\, \mu_n(dx) \\
& && \text{(partition of } Expr_\tau + \bot) \\
&= \int_{Vals_\tau + \bot} \delta_x(A)\, \mu_n(dx) + \int_{Expr_\tau \setminus Vals_\tau} K_\tau(x, A)\, \mu_n(dx) \\
& && \text{(terminal states are absorbing)} \\
&= \int_{Vals_\tau + \bot} \mathbf{1}_A(x)\, \mu_n(dx) + \int_{Expr_\tau \setminus Vals_\tau} K_\tau(x, A)\, \mu_n(dx) && (\delta_x(A) = \mathbf{1}_A(x)) \\
&= \mu_n(A) + \int_{Expr_\tau \setminus Vals_\tau} K_\tau(x, A)\, \mu_n(dx) && (A \subseteq Vals_\tau + \bot) \\
&\geq \mu_n(A) && \text{(nonnegativity)} \\
&= \langle e : \tau \rangle_n(A). && \text{(definition of } \mu_n)
\end{aligned}$$

□

Lemma F.50. *For every $e : \tau$,*

$$\langle e : \tau \rangle \in \mathcal{D}(Vals_\tau + \bot).$$

That is, $\langle e : \tau \rangle$ is a subprobability measure on $Vals_\tau + \bot$.

Proof. Fix $e : \tau$, and write

$$\mu_n := \langle e : \tau \rangle_n.$$

Each μ_n is a subprobability measure on $Expr_\tau + \bot$. Since $Vals_\tau + \bot$ is measurable, every $A \in \mathcal{F}_{Vals_\tau + \bot}$ is also measurable in $Expr_\tau + \bot$.

First,

$$\begin{aligned}\langle e : \tau\rangle(\emptyset) &= \lim_{n\to\infty} \mu_n(\emptyset) && \text{(definition of } \langle e : \tau\rangle\text{)}\\ &= \lim_{n\to\infty} 0 && (\mu_n \text{ is a measure})\\ &= 0.\end{aligned}$$

Now let

$$(A_i)_{i\in\mathbb{N}} \subseteq \mathcal{F}_{Vals_\tau+\bot}$$

be pairwise disjoint. Then

$$\begin{aligned}\langle e : \tau\rangle\left(\bigcup_{i\in\mathbb{N}} A_i\right) &= \lim_{n\to\infty} \mu_n\left(\bigcup_{i\in\mathbb{N}} A_i\right) && \text{(definition of } \langle e : \tau\rangle\text{)}\\ &= \lim_{n\to\infty} \sum_{i\in\mathbb{N}} \mu_n(A_i) && \text{(countable additivity of } \mu_n\text{)}\\ &= \sum_{i\in\mathbb{N}} \lim_{n\to\infty} \mu_n(A_i) && \text{(monotone convergence on } \mathbb{N}\text{)}\\ &= \sum_{i\in\mathbb{N}} \langle e : \tau\rangle(A_i). && \text{(definition of } \langle e : \tau\rangle\text{)}\end{aligned}$$

The monotone-convergence step applies because

$$\mu_n(A_i) \le \mu_{n+1}(A_i)$$

for every $i, n \in \mathbb{N}$, by Lemma F.49. Thus $\langle e : \tau\rangle$ is countably additive and therefore a measure on $Vals_\tau + \bot$.

Finally,

$$\begin{aligned}\langle e : \tau\rangle(Vals_\tau + \bot) &= \lim_{n\to\infty} \mu_n(Vals_\tau + \bot) && \text{(definition of } \langle e : \tau\rangle\text{)}\\ &\le 1. && (\mu_n \text{ is a subprobability measure})\end{aligned}$$

Therefore $\langle e : \tau\rangle$ is a subprobability measure on $Vals_\tau + \bot$. □

Moreover, let

$$V_\tau = \{(v : \tau, \mathcal{D}[\![v]\!] : \mathcal{D}[\![\tau]\!]) \mid v : \tau\}.$$

We equip $V_\tau + \bot$ with the subspace σ-algebra inherited from $P_\tau + \bot$.

Lemma F.51 (Monotonicity of n-step coupling semantics). *For every* $(e : \tau, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!]) \in P_\tau$, $n \in \mathbb{N}$, *and* $A \in \mathcal{F}_{V_\tau+\bot}$,

$$\langle e : \tau, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!]\rangle_n(A) \le \langle e : \tau, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!]\rangle_{n+1}(A).$$

Proof. The coupled small-step semantics maps every element of V_τ to its Dirac measure, while the $\bot$-extension maps $\bot$ to $\delta_\bot$. Thus every element of $V_\tau + \bot$ is absorbing. The remainder is analogous to the proof of Lemma F.49, replacing $Expr_\tau + \bot$ by $P_\tau + \bot$ and $Vals_\tau + \bot$ by $V_\tau + \bot$. □

Lemma F.52. *For every* $(e : \tau, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!]) \in P_\tau$,

$$\langle e : \tau, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!]\rangle \in \mathcal{D}(V_\tau + \bot).$$

That is, the coupled big-step semantics is a subprobability measure on $V_\tau + \bot$.

Proof. Analogous to the proof of Lemma F.50, replacing $Expr_\tau + \bot$ by $P_\tau + \bot$, $Vals_\tau + \bot$ by $V_\tau + \bot$, and using Lemma F.51 for monotonicity. □

Definition F.53 (Big-step semantics). Let $e : \tau$. For measurable events in the respective terminal spaces, the source and coupled *(big-step) semantics* are defined by

$$\langle e : \tau \rangle(A) := \lim_{n\to\infty} \langle e : \tau \rangle_n(A), \qquad A \in \mathcal{F}_{Vals_\tau + \bot},$$
$$\langle e : \tau, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!] \rangle(A) := \lim_{n\to\infty} \langle e : \tau, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!] \rangle_n(A), \qquad A \in \mathcal{F}_{V_\tau + \bot}.$$

By Lemmas F.49 and F.51, the source and coupled n-step semantics are monotonically increasing on every measurable event in $Vals_\tau + \bot$ and $V_\tau + \bot$, respectively. By Lemmas F.50 and F.52, their pointwise limits are subprobability measures.

Lemma F.54 (Continunity of projections). *For all $n \in \mathbb{N}$,*

(1) $\pi_1(\lim_{n\to\infty} \langle e, e' \rangle_n) = \lim_{n\to\infty} \pi_1(\langle e, e' \rangle_n)$
(2) $\pi_2(\lim_{n\to\infty} \langle e, e' \rangle_n) = \lim_{n\to\infty} \pi_2(\langle e, e' \rangle_n)$

Proof. We prove part i) in detail; the proof of part ii) is analogous. Fix an arbitrary measurable $A \in \Sigma_1$:

$$\begin{aligned}
&\pi_1(\lim_{n\to\infty} \langle e, e' \rangle_n)(A) \\
&= \left(\lim_{n\to\infty} \langle e, e' \rangle_n\right)(A \times X_2) && \text{(by definition of } \pi_1\text{)} \\
&= \sup_{n\in\mathbb{N}} \langle e, e' \rangle_n(A \times X_2) && \text{(since } \langle e, e' \rangle_n \text{ is monotone)} \\
&= \sup_{n\in\mathbb{N}} \pi_1(\langle e, e' \rangle_n)(A) && \text{(by definition of } \pi_1\text{)} \\
&= \lim_{n\to\infty} \pi_1(\langle e, e' \rangle_n)(A) && \text{(since } \pi_1(\langle e, e' \rangle_n) \text{ is monotone)}
\end{aligned}$$

Since A was arbitrary, this implies:

$$\pi_1(\lim_{n\to\infty} \langle e, e' \rangle_n) = \lim_{n\to\infty} \pi_1(\langle e, e' \rangle_n),$$

as desired. □

Lemma F.55 (n-step coupling projection). *For all $n \in \mathbb{N}$,*

(1) $\pi_1(\langle e : \tau, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!] \rangle_n) = \langle e : \tau \rangle_n$
(2) $\pi_2(\langle e : \tau, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!] \rangle_n) = \langle \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!] \rangle_n$

Proof. By induction on n. We prove i), as the proof for ii) is analogous.
Base case: $n = 0$

$$\begin{aligned}
& \pi_1(\langle e : \tau,\ \mathcal{D}[\![e']\!] : \mathcal{D}[\![\tau']\!] \rangle_0) \\
=\ & \pi_1(\delta_{(e:\tau,\ \mathcal{D}[\![e']\!] : \mathcal{D}[\![\tau']\!])}) && \text{(definition of } n\text{-step coupling semantics)} \\
=\ & \delta_{e:\tau} && \text{(projection of Dirac)} \\
=\ & \langle e : \tau \rangle_0. && \text{(definition of } n\text{-step semantics)}
\end{aligned}$$

Inductive step: $n > 0$

$$
\begin{aligned}
& \pi_1(\langle e : \tau,\ \mathcal{D}[\![e']\!] : \mathcal{D}[\![\tau']\!]\rangle_n) \\
= \ & \pi_1(\langle e : \tau, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!]\rangle_{n-1} \gg\!= \lambda(f,g).[\![f : \tau, g : \mathcal{D}[\![\tau]\!]]\!]) \\
& \qquad \text{(definition of } n\text{-step coupling semantics)} \\
= \ & (\langle e : \tau, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!]\rangle_{n-1} \gg\!= \lambda(f,g).[\![f : \tau, g : \mathcal{D}[\![\tau]\!]]\!]) \gg\!= \lambda(f,g).\delta_f \qquad \text{(by (2))} \\
= \ & \langle e : \tau, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!]\rangle_{n-1} \gg\!= \lambda(f,g).([\![f : \tau, g : \mathcal{D}[\![\tau]\!]]\!] \gg\!= \lambda(f,g).\delta_f) \qquad \text{(monad assoc.)} \\
= \ & \langle e : \tau, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!]\rangle_{n-1} \gg\!= \lambda(f,g).\pi_1([\![f,g]\!]) \qquad \text{(by (2))} \\
= \ & \langle e : \tau, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!]\rangle_{n-1} \gg\!= \lambda(f,g).[\![f]\!] \qquad \text{(Lemma F.45.1)} \\
= \ & \langle e : \tau, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!]\rangle_{n-1} \gg\!= \lambda(f,g).(\delta_f \gg\!= \lambda f.[\![f]\!]) \qquad \text{(monad unit)} \\
= \ & (\langle e : \tau, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!]\rangle_{n-1} \gg\!= \lambda(f,g).\delta_f) \gg\!= \lambda f.[\![f]\!] \qquad \text{(monad assoc.)} \\
= \ & \pi_1(\langle e : \tau, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\tau]\!]\rangle_{n-1}) \gg\!= \lambda f.[\![f]\!] \qquad \text{(by (2))} \\
= \ & \langle e : \tau\rangle_{n-1} \gg\!= \lambda f.[\![f]\!] \qquad \text{(by I.H.)} \\
= \ & \langle e : \tau\rangle_n \qquad \text{(definition of } n\text{-step semantics)}
\end{aligned}
$$

□

Before concluding the soundness proof, we relate the auxiliary statement $\mathtt{DC}_{B,B_1,B_2,V_1,V_2}$ used in the coupling semantics to the nested target expression generated by the discretization function in Figure 13.

Lemma F.56. *Suppose an expression $F(e_1, e_2)$ evaluates e_1 first, then e_2, and, when both arguments are values v_1, v_2, continues as $q(v_1, v_2)$. Then*

$$\langle F(e_1, e_2)\rangle = \langle e_1\rangle \gg\!= \lambda v_1.\langle e_2\rangle \gg\!= \lambda v_2.\langle q(v_1, v_2)\rangle.$$

All functions bound by $\gg\!=$ are extended to $\bot$ as in Definition F.29.

Proof. By induction on the n-step semantics, using the small-step rules for evaluating the first and second arguments and associativity of $\gg\!=$. Taking the limit as $n \to \infty$ gives the result. □

Theorem F.57 ($\mathtt{DC}_{B,B_1,B_2,V_1,V_2}$ is a valid construct). *Let*

$$e = \mathbf{uniform}(e_1 : \mathbf{float}[B_1; V_1], e_2 : \mathbf{float}[B_2; V_2]) : \mathbf{float}[B; \top]$$

be well typed, where $B, B_1, B_2 \neq \top$ and $V_1, V_2 \neq \top$. Let e_{case} be the nested **let***- and* **if***-expression in the last case of Figure 13. Then*

$$\langle \mathtt{DC}_{B,B_1,B_2,V_1,V_2}(\mathcal{D}[\![e_1 : \mathbf{float}[B_1; V_1]]\!], \mathcal{D}[\![e_2 : \mathbf{float}[B_2; V_2]]\!]) : \mathbf{fin}(|B|{+}1)\rangle = \langle e_{\text{case}} : \mathbf{fin}(|B|{+}1)\rangle.$$

Proof. Write

$$d_1 = \mathcal{D}[\![e_1 : \mathbf{float}[B_1; V_1]]\!] \qquad \text{and} \qquad d_2 = \mathcal{D}[\![e_2 : \mathbf{float}[B_2; V_2]]\!].$$

For arbitrary values $k_1 : \mathbf{fin}(|B_1| + 1)$ and $k_2 : \mathbf{fin}(|B_2| + 1)$, define

$$q(k_1, k_2) = \begin{cases} \mathcal{D}[\![\mathbf{uniform}(v_i, w_j) : \mathbf{float}[B; \top]]\!] & \text{if there exists } (v_i, w_j) \in V_1 \times V_2 \text{ such that} \\ & k_1 = \mathcal{D}[\![v_i : \mathbf{float}[B_1; V_1]]\!] \text{ and} \\ & k_2 = \mathcal{D}[\![w_j : \mathbf{float}[B_2; V_2]]\!], \\ \mathbf{diverge} & \text{if no such pair exists.} \end{cases}$$

When such a pair exists, it is unique because recovers(B_1, V_1) and recovers(B_2, V_2); if $V_1 = \emptyset$ or $V_2 = \emptyset$, no such pair exists and hence $q(k_1, k_2) = \mathbf{diverge}$ for all k_1, k_2.

By the operational rule for $\mathtt{DC}_{B,B_1,B_2,V_1,V_2}$ and Lemma F.56,

$$\langle \mathtt{DC}_{B,B_1,B_2,V_1,V_2}(d_1,d_2)\rangle = \langle d_1\rangle \gg\!= \lambda k_1.\langle d_2\rangle \gg\!= \lambda k_2.\langle q(k_1,k_2)\rangle. \tag{1}$$

Now consider e_{case}. Its two outer **let**-bindings evaluate d_1 and d_2 in the same order. Once they evaluate to k_1 and k_2, finite equality and conjunction select the unique branch indexed by (v_i, w_j) when a matching pair exists; otherwise, including whenever $V_1 = \emptyset$ or $V_2 = \emptyset$, the final branch yields **diverge**. Thus the remaining nested conditional evaluates to $q(k_1, k_2)$. Applying Lemma F.56 again gives

$$\langle e_{\text{case}}\rangle = \langle d_1\rangle \gg\!= \lambda k_1.\langle d_2\rangle \gg\!= \lambda k_2.\langle q(k_1,k_2)\rangle. \tag{2}$$

The right-hand sides of (1) and (2) are identical. Therefore,

$$\langle \mathtt{DC}_{B,B_1,B_2,V_1,V_2}(d_1,d_2)\rangle = \langle e_{\text{case}}\rangle. \qquad \square$$

The following will imply our desired equivalence claim:

Lemma F.58 (Big-step coupling projection). *Let* $\langle e : \tau, \mathcal{D}[\![e']\!] : \mathcal{D}[\![\tau']\!]\rangle \in \mathcal{D}(P_\tau + \bot)$. *Then,*

(1) $\pi_1(\langle e : \tau, \mathcal{D}[\![e']\!] : \mathcal{D}[\![\tau']\!]\rangle) = \langle e : \tau\rangle$
(2) $\pi_2(\langle e : \tau, \mathcal{D}[\![e']\!] : \mathcal{D}[\![\tau']\!]\rangle) = \langle \mathcal{D}[\![e']\!] : \mathcal{D}[\![\tau']\!]\rangle$

Proof.

$$\begin{aligned}
&\pi_1(\langle e : \tau, \mathcal{D}[\![e']\!] : \mathcal{D}[\![\tau']\!]\rangle) \\
&= \pi_1(\lim_{n\to\infty} \langle e : \tau, \mathcal{D}[\![e']\!] : \mathcal{D}[\![\tau']\!]\rangle_n) && \text{(by defintion of big-step coupling semantics)} \\
&= \lim_{n\to\infty} \pi_1(\langle e : \tau, \mathcal{D}[\![e']\!] : \mathcal{D}[\![\tau']\!]\rangle_n) && \text{(by communitivity lemma)} \\
&= \lim_{n\to\infty} \langle e : \tau\rangle_n && \text{(by definition of n-step projection lemma)} \\
&= \langle e : \tau\rangle && \text{(by definition of big-step semantics)}
\end{aligned}$$

$\square$

Theorem F.59. *Let* $e : \mathbf{bool}$. *Then* $\langle e : \mathbf{bool}\rangle = \langle \mathcal{D}[\![e : \mathbf{bool}]\!] : \mathbf{bool}\rangle$.

Proof. Since $\mathcal{D}(Vals_\tau + \bot)$ contains only discrete distributions, we proceed by case distinction on the probability of the final outcome $v \in \{\text{true}, \text{false}, \bot\}$.

The case $v = \text{true}$. We have

$$\begin{aligned}
&\langle e : \mathbf{bool}\rangle(\text{true}) \\
&= \pi_1(\langle e : \mathbf{bool}, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\mathbf{bool}]\!]\rangle)(\text{true}) && \text{(Lemma F.58.1)} \\
&= (\langle e : \mathbf{bool}, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\mathbf{bool}]\!]\rangle \gg\!= \lambda(e_1,e_2).\delta_{e_1})(\text{true}) && \text{(definition)} \\
&= \langle e : \mathbf{bool}, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\mathbf{bool}]\!]\rangle((\text{true},\text{true})) \cdot \delta_{\text{true}}(\text{true}) \\
&\quad + \langle e : \mathbf{bool}, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\mathbf{bool}]\!]\rangle((\text{false},\text{false})) \cdot \delta_{\text{false}}(\text{true}) \\
&\quad + \langle e : \mathbf{bool}, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\mathbf{bool}]\!]\rangle(\bot) \cdot \delta_\bot(\text{true}) && \text{(definition of } \gg\!=\text{)} \\
&= \langle e : \mathbf{bool}, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\mathbf{bool}]\!]\rangle((\text{true},\text{true})) && \text{(definition of } \delta\text{)} \\
&= \pi_2(\langle e : \mathbf{bool}, \mathcal{D}[\![e]\!] : \mathcal{D}[\![\mathbf{bool}]\!]\rangle)(\text{true}) && \text{(reasoning completely analogous)} \\
&= \langle \mathcal{D}[\![e]\!] : \mathcal{D}[\![\mathbf{bool}]\!]\rangle(\text{true})\ . && \text{(Lemma F.58.2)}
\end{aligned}$$

The cases $v = \text{false}$ and $v = \bot$ are completely analogous.

$\square$

G Benchmarks

G.1 Asymptotic Scaling

G.1.1 Code Sketches for Scaling Benchmarks. This section contains code snippets of the programs (Figures 29a to 29d and 30a to 30d) used in the scaling experiments in §7.2. We implemented a Python script that automatically generates these programs based on configurable parameters such as program size, guard structure, and dependency pattern. For reproducibility, we used a fixed random seed to generate each program. All r_i values in the generated programs, where $0 \leq r_i \leq 1$ is randomly chosen, are rounded to two decimal places.

All of these programs follow the same structural pattern: they define a sequence of random variables $X_1, \ldots, X_n$, where each X_i is drawn from a uniform distribution whose parameters depend on the value of a single predecessor variable through a conditional branch of the form:

$$X_i = \begin{cases} \text{Unif}(a_i, b_i), & X_{\pi(i)} < r_i, \\ \text{Unif}(a'_i, b'_i), & \text{otherwise}, \end{cases}$$

for some parent selection function $\pi : \{2, \ldots, n\} \to \{1, \ldots, i-1\}$. Thus each program is a Bayesian network of conditionally independent nodes, differing only in the choice of $\pi(i)$ that determines the dependency pattern.

The code snippets show representative size 5 instances from each benchmark family. The scaling experiments generate larger instances of the same form by varying the program size and, for the alternating benchmarks, the guard span. The programs in Figure 29a follow the scheme depicted above where the program size is n, i.e., the number of comparisons performed is n. In Figure 29b, each x_i does not necessarily depend on x_{i-1} but on some randomly chosen predecessor[3]. In Figures 29c and 29d, we additionally choose at random whether the guards of the conditionals and the branches are either some predecessor variable x_i or a **`uniform`**-sampling instruction. The programs in Figure 30a also follow the scheme depicted above but where the variables used in the guards cycle with a fixed span size of m. For instance, for $m = 2$, the pattern is $x_1, x_2, x_1, \ldots$. The programs in Figures 30b to 30d follow a similar pattern.

We report runtime (seconds) for (i) Sppl, (ii) Slice (discretization only), (iii) Slice+Dice (discretization + inference via Dice), and (iv) Slice+Roulette (discretization + inference via Roulette). The timeout is 120 seconds.

Results. Discretization remains inexpensive across these benchmark families, and the resulting Slice+Dice/Roulette pipelines scale well with increasing program size. In contrast, exact inference with Sppl exhibits substantially steeper runtime growth and reaches the timeout on several benchmark variants.

G.2 Decision Trees

Tables 5 and 6 report the decision tree benchmarks for varying tree sizes n. The first table contains the original decision tree benchmarks from [1], while the second extends this evaluation with additional, larger instances that we generated to more systematically study scaling as the tree size increases.

Results. We observe that Slice+Dice achieves the lowest total runtime in practically all DT programs. Slice+Roulette is competitive only on very small trees, and becomes substantially slower by DT_{100}–DT_{500} and times out from DT_{800} onward. Additionally, the discretization cost grows substantially by DT_{1000}–DT_{5000}. Psi completes the three DT_4 variants and the independent DT_{14}

[3]We generate these programs in a randomized manner with a fixed seed for reproducibility

```
let x1 = uniform(0,1) in
let x2 = if x1 < 0.56 then uniform(0,2) else uniform(0,3) in
let x3 = if x2 < 0.92 then uniform(0,4) else uniform(0,5) in
let x4 = if x3 < 0.47 then uniform(0,6) else uniform(0,7) in
let x5 = if x4 < 0.51 then uniform(0,8) else uniform(0,9) in
x5 < 0.5
```

(a) Conditional Independent (Figure 31a): Size 5 chain in which each variable depends on its immediate predecessor; all uniform distributions are distinct.

```
let x1 = uniform(0,1) in
let x2 = if x1 < 0.56 then uniform(0,2) else uniform(0,3) in
let x3 = if x1 < 0.45 then uniform(0,4) else uniform(0,5) in
let x4 = if x1 < 0.18 then uniform(0,6) else uniform(0,7) in
let x5 = if x1 < 0.48 then uniform(0,8) else uniform(0,9) in
x5 < 0.5
```

(b) Conditional Independent Fork (Figure 31b): Size 5 fork in which x_1 is the sole parent of all subsequent variables; all uniform distributions are distinct.

```
let x1 = uniform(0,1) in
let x2 = if x1 < 0.18 then uniform(0,2) else uniform(0,3) in
let x3 = if x1 < 0.63 then uniform(0,4) else uniform(0,5) in
let x4 = if x2 < 0.09 then uniform(0,6) else uniform(0,7) in
let x5 = if x1 < 0.14 then uniform(0,8) else uniform(0,9) in
x5 < 0.5
```

(c) Conditional Random 1 (Figure 31c): Size 5 program with randomly selected predecessors. Some variables may be unused; all uniform distributions are distinct.

```
let x1 = uniform(0, 10) in
let x2 = if x1 < 0.06 then x1 else x1 in
let x3 = if uniform(0, 4) < 0.64 then x2 else x1 in
let x4 = if x3 < 0.71 then uniform(0, 5) else x1 in
let x5 = if uniform(0, 4) < 0.4 then x1 else uniform(0, 2) in
x4 < 0.5
```

(d) Conditional Random 2 (Figure 31d): Size 5 program with randomized guards and branches. Uniform upper bounds are random integers from 0 to 10 and may repeat.

Fig. 29. Representative programs from the scaling benchmarks.

variant, but times out on the remaining programs. Hakaru's disintegration phase remains fast, but its subsequent simplification is less robust: it produces errors on several smaller independent variants, approaches the timeout at DT_{500}, and times out on all variants from DT_{800} onward.

```
let x1 = uniform(0,1) in
let x2 = if x1 < 0.02 then uniform(0,2) else uniform(0,3) in
let x3 = if x2 < 0.68 then uniform(0,4) else uniform(0,5) in
let x4 = if x1 < 0.21 then uniform(0,6) else uniform(0,7) in
let x5 = if x2 < 0.93 then uniform(0,8) else uniform(0,9) in
x5 < 0.5
```

(a) Alternating Guard 1 (Figure 32a): Size 5 program with guard span 2. Guard variables cycle over the span, so variables beyond it may be unused.

```
let x1 = uniform(0,1) in
let x2 = if x1 < 0.38 then uniform(0,2) else x1 in
let x3 = if x2 < 0.71 then uniform(0,4) else x2 in
let x4 = if x1 < 0.42 then uniform(0,6) else x3 in
let x5 = if x2 < 0.57 then uniform(0,8) else x4 in
x5 < 0.5
```

(b) Alternating Guard 2 (Figure 32b): Size 5 program with guard span 2. Every variable is later used as a guard or else-branch expression; all uniform distributions are distinct.

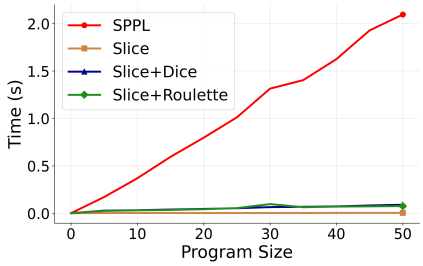
```
let x1 = uniform(0,1) in
let x2 = if x1 < 0.67 then uniform(0,2) else uniform(0,3) in
let x3 = if x2 < 0.5 then x2 else uniform(0,4) in
let x4 = if x1 < 0.5 then x3 else uniform(0,6) in
let x5 = if x2 < 0.5 then x4 else uniform(0,8) in
x5 < 0.5
```

(c) Alternating Guard 3 (Figure 32c): Size 5 program with guard span 2. Every variable is later used as a guard or then-branch expression; all uniform distributions are distinct.

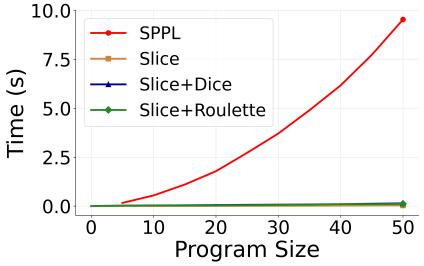
```
let x1 = uniform(0,2) in
let x2 = if x1 < 0.13 then uniform(0,2) else uniform(0,1) in
let x3 = if x2 < 0.06 then uniform(0,8) else uniform(0,3) in
let x4 = if x1 < 0.68 then uniform(0,4) else uniform(0,8) in
let x5 = if x2 < 0.51 then uniform(0,3) else uniform(0,7) in
x5 < 0.5
```

(d) Random Alternating (Figure 32d): Size 5 program with guard span 2. Uniform upper bounds are random integers from 0 to 10 and may repeat.

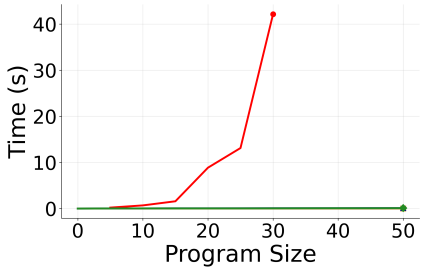
Fig. 30. Representative programs from the scaling benchmarks.



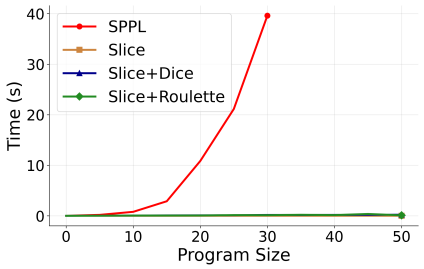
(a) Conditional Independent



(b) Conditional Independent – Fork

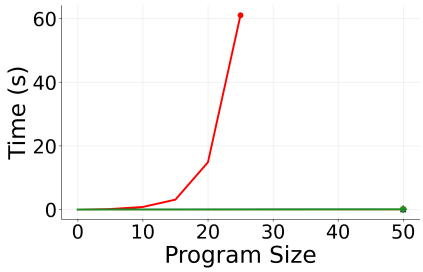


(c) Conditional Independent – Rand. 1

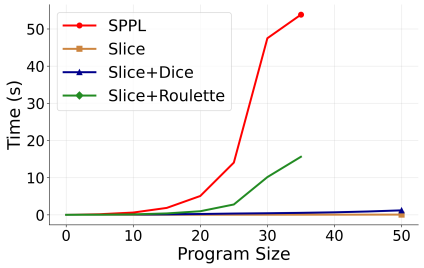


(d) Conditional Independent – Rand. 2

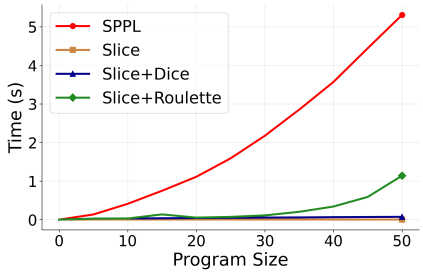
Fig. 31. Scaling graphs for conditionally independent programs. Time-out=120s; a disappearing line indicates timeout. Graphs (a)–(d) correspond, respectively, to the representative programs in Figures 29a to 29d.



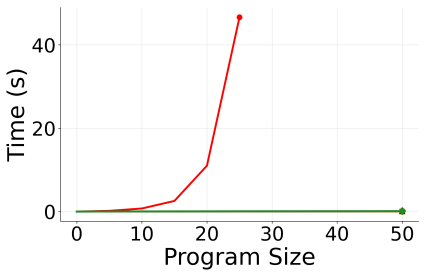
(a) Alt. Guard 1



(b) Alt. Guard 2



(c) Alt. Guard 3



(d) Alt. Guard – Rand.

Fig. 32. Scaling graphs for conditionally independent programs with alternating guards. Time-out=120s; a disappearing line indicates timeout. Graphs (a)–(d) correspond, respectively, to the representative programs in Figures 30a to 30d.

Table 5. Execution times (seconds) for the benchmarks from [1], with a timeout of 120s. The `err.`'s from Hakaru's simplification step represent a crash, caused by an internal symbolic processing exception.

Decision Program	Population Model	Slice+Dice (s)			Slice+Roulette (s)			SPPL (s)			PSI (s)	Hakaru (s)		
		Slice	Dice	Total	Slice	Roulette	Total	Trans.	Judg.	Total		Disint.	Simplify	Total
DT_4	Ind	0.002	0.031	0.033	0.002	0.026	**0.028**	0.074	0.011	0.085	3.477	0.104	1.806	1.910
	BN1	0.002	0.033	0.035	0.002	0.031	**0.033**	0.245	0.028	0.273	19.890	0.031	0.426	0.457
	BN2	0.002	0.035	**0.037**	0.003	0.034	**0.037**	0.268	0.030	0.298	51.107	0.030	0.528	0.558
DT_{14}	Ind	0.002	0.047	**0.049**	0.002	0.049	0.051	0.153	0.024	0.177	26.150	0.038	err.	err.
	BN1	0.002	0.056	**0.058**	0.002	0.074	0.076	0.597	0.076	0.673	t.o.	0.029	1.067	1.096
	BN2	0.002	0.058	**0.060**	0.002	0.073	0.075	0.535	0.072	0.607	t.o.	0.028	2.619	2.647
DT_{16}	Ind	0.002	0.045	**0.047**	0.002	0.054	0.056	0.176	0.026	0.202	t.o.	0.031	err.	err.
	BN1	0.002	0.053	**0.055**	0.002	0.079	0.081	0.535	0.088	0.623	t.o.	0.036	1.125	1.161
	BN2	0.003	0.053	**0.056**	0.003	0.079	0.082	0.554	0.090	0.644	t.o.	0.035	1.594	1.629
DT^{α}_{16}	Ind	0.002	0.034	**0.036**	0.002	0.052	0.054	0.201	0.029	0.230	t.o.	0.031	err.	err.
	BN1	0.002	0.042	**0.044**	0.003	0.191	0.194	0.597	0.080	0.677	t.o.	0.036	err.	err.
	BN2	0.003	0.049	**0.052**	0.003	0.224	0.227	0.642	0.075	0.717	t.o.	0.040	2.142	2.182
DT_{44}	Ind	0.002	0.099	**0.101**	0.002	0.276	0.278	0.402	0.073	0.475	t.o.	0.041	t.o.	t.o.
	BN1	0.003	0.120	**0.123**	0.003	0.550	0.553	1.043	0.309	1.352	t.o.	0.042	15.972	16.014
	BN2	0.003	0.122	**0.125**	0.003	0.546	0.549	1.135	0.346	1.481	t.o.	0.053	44.456	44.509

Table 6. Execution times (seconds) for the benchmarks from [1], with a timeout of 120s. For DT_{5000} under our setup, SPPL returned a maximum recursion depth exceeded error (reported as `err.`).

Decision Program	Population Model	Slice+Dice (s)			Slice+Roulette (s)			SPPL (s)			PSI (s)	Hakaru (s)		
		Slice	Dice	Total	Slice	Roulette	Total	Trans.	Judg.	Total		Disint.	Simplify	Total
DT_{50}	Ind	0.006	0.094	**0.100**	0.004	0.156	0.160	0.765	0.029	0.793	t.o.	0.045	3.712	3.757
	BN1	0.007	0.079	**0.086**	0.006	0.451	0.457	1.646	0.085	1.731	t.o.	0.039	3.548	3.587
	BN2	0.007	0.083	**0.090**	0.006	0.403	0.409	1.580	0.080	1.660	t.o.	0.037	3.635	3.672
DT_{100}	Ind	0.011	0.138	**0.149**	0.011	0.654	0.665	1.455	0.036	1.490	t.o.	0.057	8.490	8.547
	BN1	0.015	0.148	**0.163**	0.016	1.775	1.791	2.867	0.150	3.017	t.o.	0.077	8.596	8.673
	BN2	0.017	0.153	**0.170**	0.016	1.775	1.791	2.914	0.152	3.066	t.o.	0.069	8.816	8.885
DT_{200}	Ind	0.045	0.289	**0.334**	0.045	4.089	4.134	2.710	0.059	2.770	t.o.	0.099	24.551	24.650
	BN1	0.064	0.322	**0.386**	0.064	9.496	9.560	5.836	0.318	6.154	t.o.	0.111	23.741	23.852
	BN2	0.068	0.305	**0.373**	0.068	9.556	9.624	5.797	0.302	6.099	t.o.	0.115	23.988	24.103
DT_{500}	Ind	0.479	0.879	**1.358**	0.515	55.677	56.192	7.908	0.141	8.049	t.o.	0.295	118.438	118.733
	BN1	0.599	0.961	**1.560**	0.636	113.426	114.062	15.017	0.787	15.804	t.o.	0.293	117.507	117.800
	BN2	0.616	0.999	**1.615**	0.610	112.976	113.586	15.330	0.889	16.219	t.o.	0.301	113.667	113.968
DT_{800}	Ind	1.738	1.592	**3.330**	1.738	t.o.	t.o.	14.698	0.238	14.937	t.o.	0.488	t.o.	t.o.
	BN1	2.184	1.690	**3.874**	2.184	t.o.	t.o.	26.297	1.265	27.561	t.o.	0.597	t.o.	t.o.
	BN2	2.151	1.704	**3.855**	2.151	t.o.	t.o.	26.542	1.298	27.840	t.o.	0.512	t.o.	t.o.
DT_{1000}	Ind	3.584	2.096	**5.680**	3.584	t.o.	t.o.	17.626	0.279	17.905	t.o.	0.774	t.o.	t.o.
	BN1	4.125	2.192	**6.317**	4.125	t.o.	t.o.	33.417	1.537	34.954	t.o.	0.784	t.o.	t.o.
	BN2	4.052	2.152	**6.204**	4.052	t.o.	t.o.	34.225	1.612	35.837	t.o.	0.803	t.o.	t.o.
DT_{2000}	Ind	26.791	4.607	**31.398**	26.791	t.o.	t.o.	44.957	0.786	45.742	t.o.	2.868	t.o.	t.o.
	BN1	29.148	4.726	**33.874**	29.148	t.o.	t.o.	84.744	3.227	87.972	t.o.	2.775	t.o.	t.o.
	BN2	30.353	4.715	**35.068**	30.353	t.o.	t.o.	76.398	3.152	79.550	t.o.	2.944	t.o.	t.o.
DT_{5000}	Ind	t.o.	t.o.	**t.o.**	t.o.	t.o.	**t.o.**	err.	err.	**err.**	t.o.	16.030	t.o.	t.o.
	BN1	t.o.	t.o.	**t.o.**	t.o.	t.o.	**t.o.**	err.	err.	**err.**	t.o.	14.422	t.o.	t.o.
	BN2	t.o.	t.o.	**t.o.**	t.o.	t.o.	**t.o.**	err.	err.	**err.**	t.o.	14.933	t.o.	t.o.

G.3 Baselines

We finally evaluate Slice on benchmarks drawn from prior work [21, 40]. Most benchmark programs are taken directly from the Sppl Github repository and translated into Slice. In some cases, the Sppl versions differ from the original Psi benchmarks, for example by replacing continuous choices with finite ones so that the programs fall within the class supported by Sppl. CoinBias and SurveyUnbias are not included in the Sppl repository, so we adapted these benchmarks in the same style. Consequently, the Slice and Sppl columns in Table 7 evaluate the same benchmark programs and provide an apples-to-apples comparison. The full benchmark programs are provided in §G.3.1. The first column in the table gives the benchmark name, the second shows the number of datasets on which to condition the program, and the remaining columns display execution times in seconds. For benchmarks with both modified and unmodified versions, the Psi and Hakaru cells report the modified timing marked by • above the unmodified timing marked by ○. The • entries therefore provide the apples-to-apples comparison with Slice and Sppl. For Hakaru, we report disintegration, subsequent simplification, and their total. `t.o.` denotes the same 120s timeout used elsewhere, and `err.` denotes a tool error.

Table 7. Execution times (in seconds) for benchmarks from PSI [21] and SPPL [40], with modified and unmodified Psi and Hakaru exact inference baselines. Timeout=120s. For benchmarks with two outputs for Psi and Hakaru, • marks the modified program and ○ marks the unmodified program. `err.` represents a crash, caused by out of memory.

Benchmark	Datasets	Slice+Dice (s)	Slice+Roulette (s)	SPPL (s)	PSI (s)	Hakaru (s)		
						Disintegrate	Simplify	Total
ClickGraph [21]	10	**1.469**	t.o.	t.o.	t.o. • t.o. ○	0.046 • 0.035 ○	t.o. • 11.287 ○	t.o. • 11.322 ○
ClinicalTrial [21]	20	**1.582**	5.020	17.895	t.o. • 1.916 ○	t.o. • t.o. ○	t.o. • t.o. ○	t.o. • t.o. ○
CoinBias [21]	5	**0.466**	0.590	2.382	20.138 • 0.369 ○	0.058 • 0.025 ○	4.789 • 0.528 ○	4.847 • 0.553 ○
Election [40]	–	**1.587**	t.o.	19.410	t.o.	0.019	t.o.	t.o.
MarkovSwitching [40]	–	0.423	**0.240**	19.998	t.o.	0.020	0.275	0.295
StudentInterviews [40]	–	3.331	t.o.	53.209	t.o.	0.025	0.443	**0.468**
SurveyUnbias [21]	5	75.221	**0.651**	3.519	t.o. • t.o. ○	0.027 • 0.029 ○	t.o. • err. ○	t.o. • err. ○
TrueSkill [21]	3	**0.750**	68.669	32.881	t.o. • 0.068 ○	0.027 • 0.018 ○	err. • 0.215 ○	err. • 0.233 ○

Results. Across the modified baseline programs in Table 7, the Slice pipelines generally provide the fastest successful end-to-end inference: Slice+Dice performs best on most benchmarks, while Slice+Roulette is faster on some instances. Sppl is generally slower, and Psi frequently times out, although both Psi and Hakaru perform better on some unmodified programs. Hakaru's disintegration phase is usually fast, but subsequent Maple simplification often times out or returns an error, limiting its end-to-end reliability.

G.3.1 Baseline Programs. The exact Slice programs used for these baseline benchmarks are reproduced below.

ClickGraph

```
let p_similar = beta(1, 1) in

let similar_0 =
    if p_similar <= 0.05 then discrete(0.95, 0.05)
    else if p_similar <= 0.1 then discrete(0.9, 0.1)
    else if p_similar <= 0.15 then discrete(0.85, 0.15)
    else if p_similar <= 0.2 then discrete(0.8, 0.2)
    else if p_similar <= 0.25 then discrete(0.75, 0.25)
    else if p_similar <= 0.3 then discrete(0.7, 0.3)
    else if p_similar <= 0.35 then discrete(0.65, 0.35)
    else if p_similar <= 0.4 then discrete(0.6, 0.4)
    else if p_similar <= 0.45 then discrete(0.55, 0.45)
    else if p_similar <= 0.5 then discrete(0.5, 0.5)
    else if p_similar <= 0.55 then discrete(0.45, 0.55)
    else if p_similar <= 0.6 then discrete(0.4, 0.6)
    else if p_similar <= 0.65 then discrete(0.35, 0.65)
    else if p_similar <= 0.7 then discrete(0.3, 0.7)
    else if p_similar <= 0.75 then discrete(0.25, 0.75)
    else if p_similar <= 0.8 then discrete(0.2, 0.8)
    else if p_similar <= 0.85 then discrete(0.15, 0.85)
    else if p_similar <= 0.9 then discrete(0.1, 0.9)
    else if p_similar <= 0.95 then discrete(0.05, 0.95)
    else discrete(0.0, 1.0)
in

let p_click_url_A_0 =
    if similar_0 ==#2 1#2 then uniform(0,1) else uniform(0, 1)
in

let p_click_url_B_0 =
    if similar_0 ==#2 1#2 then p_click_url_A_0 else uniform(0, 1)
in

let click_url_A_0 =
    if p_click_url_A_0 <= 0.1 then discrete(0.9, 0.1)
    else if p_click_url_A_0 <= 0.2 then discrete(0.8, 0.2)
    else if p_click_url_A_0 <= 0.3 then discrete(0.7, 0.3)
    else if p_click_url_A_0 <= 0.4 then discrete(0.6, 0.4)
    else if p_click_url_A_0 <= 0.5 then discrete(0.5, 0.5)
    else if p_click_url_A_0 <= 0.6 then discrete(0.4, 0.6)
    else if p_click_url_A_0 <= 0.7 then discrete(0.3, 0.7)
    else if p_click_url_A_0 <= 0.8 then discrete(0.2, 0.8)
    else if p_click_url_A_0 <= 0.9 then discrete(0.1, 0.9)
    else discrete(0.0, 1.0)
in

let click_url_B_0 =
    if p_click_url_B_0 <= 0.1 then discrete(0.9, 0.1)
    else if p_click_url_B_0 <= 0.2 then discrete(0.8, 0.2)
    else if p_click_url_B_0 <= 0.3 then discrete(0.7, 0.3)
    else if p_click_url_B_0 <= 0.4 then discrete(0.6, 0.4)
    else if p_click_url_B_0 <= 0.5 then discrete(0.5, 0.5)
    else if p_click_url_B_0 <= 0.6 then discrete(0.4, 0.6)
    else if p_click_url_B_0 <= 0.7 then discrete(0.3, 0.7)
    else if p_click_url_B_0 <= 0.8 then discrete(0.2, 0.8)
    else if p_click_url_B_0 <= 0.9 then discrete(0.1, 0.9)
    else discrete(0.0, 1.0)
in

let similar_1 =
    if p_similar <= 0.05 then discrete(0.95, 0.05)
    else if p_similar <= 0.1 then discrete(0.9, 0.1)
    else if p_similar <= 0.15 then discrete(0.85, 0.15)
    else if p_similar <= 0.2 then discrete(0.8, 0.2)
    else if p_similar <= 0.25 then discrete(0.75, 0.25)
    else if p_similar <= 0.3 then discrete(0.7, 0.3)
```

```
    else if p_similar <= 0.35 then discrete(0.65, 0.35)
    else if p_similar <= 0.4 then discrete(0.6, 0.4)
    else if p_similar <= 0.45 then discrete(0.55, 0.45)
    else if p_similar <= 0.5 then discrete(0.5, 0.5)
    else if p_similar <= 0.55 then discrete(0.45, 0.55)
    else if p_similar <= 0.6 then discrete(0.4, 0.6)
    else if p_similar <= 0.65 then discrete(0.35, 0.65)
    else if p_similar <= 0.7 then discrete(0.3, 0.7)
    else if p_similar <= 0.75 then discrete(0.25, 0.75)
    else if p_similar <= 0.8 then discrete(0.2, 0.8)
    else if p_similar <= 0.85 then discrete(0.15, 0.85)
    else if p_similar <= 0.9 then discrete(0.1, 0.9)
    else if p_similar <= 0.95 then discrete(0.05, 0.95)
    else discrete(0.0, 1.0)
in

let p_click_url_A_1 =
    if similar_1 ==#2 1#2 then uniform(0,1) else uniform(0, 1)
in

let p_click_url_B_1 =
    if similar_1 ==#2 1#2 then p_click_url_A_1 else uniform(0, 1)
in

let click_url_A_1 =
    if p_click_url_A_1 <= 0.1 then discrete(0.9, 0.1)
    else if p_click_url_A_1 <= 0.2 then discrete(0.8, 0.2)
    else if p_click_url_A_1 <= 0.3 then discrete(0.7, 0.3)
    else if p_click_url_A_1 <= 0.4 then discrete(0.6, 0.4)
    else if p_click_url_A_1 <= 0.5 then discrete(0.5, 0.5)
    else if p_click_url_A_1 <= 0.6 then discrete(0.4, 0.6)
    else if p_click_url_A_1 <= 0.7 then discrete(0.3, 0.7)
    else if p_click_url_A_1 <= 0.8 then discrete(0.2, 0.8)
    else if p_click_url_A_1 <= 0.9 then discrete(0.1, 0.9)
    else discrete(0.0, 1.0)
in

let click_url_B_1 =
    if p_click_url_B_1 <= 0.1 then discrete(0.9, 0.1)
    else if p_click_url_B_1 <= 0.2 then discrete(0.8, 0.2)
    else if p_click_url_B_1 <= 0.3 then discrete(0.7, 0.3)
    else if p_click_url_B_1 <= 0.4 then discrete(0.6, 0.4)
    else if p_click_url_B_1 <= 0.5 then discrete(0.5, 0.5)
    else if p_click_url_B_1 <= 0.6 then discrete(0.4, 0.6)
    else if p_click_url_B_1 <= 0.7 then discrete(0.3, 0.7)
    else if p_click_url_B_1 <= 0.8 then discrete(0.2, 0.8)
    else if p_click_url_B_1 <= 0.9 then discrete(0.1, 0.9)
    else discrete(0.0, 1.0)
in

let similar_2 =
    if p_similar <= 0.05 then discrete(0.95, 0.05)
    else if p_similar <= 0.1 then discrete(0.9, 0.1)
    else if p_similar <= 0.15 then discrete(0.85, 0.15)
    else if p_similar <= 0.2 then discrete(0.8, 0.2)
    else if p_similar <= 0.25 then discrete(0.75, 0.25)
    else if p_similar <= 0.3 then discrete(0.7, 0.3)
    else if p_similar <= 0.35 then discrete(0.65, 0.35)
    else if p_similar <= 0.4 then discrete(0.6, 0.4)
    else if p_similar <= 0.45 then discrete(0.55, 0.45)
    else if p_similar <= 0.5 then discrete(0.5, 0.5)
    else if p_similar <= 0.55 then discrete(0.45, 0.55)
    else if p_similar <= 0.6 then discrete(0.4, 0.6)
    else if p_similar <= 0.65 then discrete(0.35, 0.65)
    else if p_similar <= 0.7 then discrete(0.3, 0.7)
    else if p_similar <= 0.75 then discrete(0.25, 0.75)
    else if p_similar <= 0.8 then discrete(0.2, 0.8)
    else if p_similar <= 0.85 then discrete(0.15, 0.85)
    else if p_similar <= 0.9 then discrete(0.1, 0.9)
    else if p_similar <= 0.95 then discrete(0.05, 0.95)
    else discrete(0.0, 1.0)
```

```
in

let p_click_url_A_2 =
    if similar_2 ==#2 1#2 then uniform(0,1) else uniform(0, 1)
in

let p_click_url_B_2 =
    if similar_2 ==#2 1#2 then p_click_url_A_2 else uniform(0, 1)
in

let click_url_A_2 =
    if p_click_url_A_2 <= 0.1 then discrete(0.9, 0.1)
    else if p_click_url_A_2 <= 0.2 then discrete(0.8, 0.2)
    else if p_click_url_A_2 <= 0.3 then discrete(0.7, 0.3)
    else if p_click_url_A_2 <= 0.4 then discrete(0.6, 0.4)
    else if p_click_url_A_2 <= 0.5 then discrete(0.5, 0.5)
    else if p_click_url_A_2 <= 0.6 then discrete(0.4, 0.6)
    else if p_click_url_A_2 <= 0.7 then discrete(0.3, 0.7)
    else if p_click_url_A_2 <= 0.8 then discrete(0.2, 0.8)
    else if p_click_url_A_2 <= 0.9 then discrete(0.1, 0.9)
    else discrete(0.0, 1.0)
in

let click_url_B_2 =
    if p_click_url_B_2 <= 0.1 then discrete(0.9, 0.1)
    else if p_click_url_B_2 <= 0.2 then discrete(0.8, 0.2)
    else if p_click_url_B_2 <= 0.3 then discrete(0.7, 0.3)
    else if p_click_url_B_2 <= 0.4 then discrete(0.6, 0.4)
    else if p_click_url_B_2 <= 0.5 then discrete(0.5, 0.5)
    else if p_click_url_B_2 <= 0.6 then discrete(0.4, 0.6)
    else if p_click_url_B_2 <= 0.7 then discrete(0.3, 0.7)
    else if p_click_url_B_2 <= 0.8 then discrete(0.2, 0.8)
    else if p_click_url_B_2 <= 0.9 then discrete(0.1, 0.9)
    else discrete(0.0, 1.0)
in

let similar_3 =
    if p_similar <= 0.05 then discrete(0.95, 0.05)
    else if p_similar <= 0.1 then discrete(0.9, 0.1)
    else if p_similar <= 0.15 then discrete(0.85, 0.15)
    else if p_similar <= 0.2 then discrete(0.8, 0.2)
    else if p_similar <= 0.25 then discrete(0.75, 0.25)
    else if p_similar <= 0.3 then discrete(0.7, 0.3)
    else if p_similar <= 0.35 then discrete(0.65, 0.35)
    else if p_similar <= 0.4 then discrete(0.6, 0.4)
    else if p_similar <= 0.45 then discrete(0.55, 0.45)
    else if p_similar <= 0.5 then discrete(0.5, 0.5)
    else if p_similar <= 0.55 then discrete(0.45, 0.55)
    else if p_similar <= 0.6 then discrete(0.4, 0.6)
    else if p_similar <= 0.65 then discrete(0.35, 0.65)
    else if p_similar <= 0.7 then discrete(0.3, 0.7)
    else if p_similar <= 0.75 then discrete(0.25, 0.75)
    else if p_similar <= 0.8 then discrete(0.2, 0.8)
    else if p_similar <= 0.85 then discrete(0.15, 0.85)
    else if p_similar <= 0.9 then discrete(0.1, 0.9)
    else if p_similar <= 0.95 then discrete(0.05, 0.95)
    else discrete(0.0, 1.0)
in

let p_click_url_A_3 =
    if similar_3 ==#2 1#2 then uniform(0,1) else uniform(0, 1)
in

let p_click_url_B_3 =
    if similar_3 ==#2 1#2 then p_click_url_A_3 else uniform(0, 1)
in

let click_url_A_3 =
    if p_click_url_A_3 <= 0.1 then discrete(0.9, 0.1)
    else if p_click_url_A_3 <= 0.2 then discrete(0.8, 0.2)
    else if p_click_url_A_3 <= 0.3 then discrete(0.7, 0.3)
```

```
    else if p_click_url_A_3 <= 0.4 then discrete(0.6, 0.4)
    else if p_click_url_A_3 <= 0.5 then discrete(0.5, 0.5)
    else if p_click_url_A_3 <= 0.6 then discrete(0.4, 0.6)
    else if p_click_url_A_3 <= 0.7 then discrete(0.3, 0.7)
    else if p_click_url_A_3 <= 0.8 then discrete(0.2, 0.8)
    else if p_click_url_A_3 <= 0.9 then discrete(0.1, 0.9)
    else discrete(0.0, 1.0)
in

let click_url_B_3 =
    if p_click_url_B_3 <= 0.1 then discrete(0.9, 0.1)
    else if p_click_url_B_3 <= 0.2 then discrete(0.8, 0.2)
    else if p_click_url_B_3 <= 0.3 then discrete(0.7, 0.3)
    else if p_click_url_B_3 <= 0.4 then discrete(0.6, 0.4)
    else if p_click_url_B_3 <= 0.5 then discrete(0.5, 0.5)
    else if p_click_url_B_3 <= 0.6 then discrete(0.4, 0.6)
    else if p_click_url_B_3 <= 0.7 then discrete(0.3, 0.7)
    else if p_click_url_B_3 <= 0.8 then discrete(0.2, 0.8)
    else if p_click_url_B_3 <= 0.9 then discrete(0.1, 0.9)
    else discrete(0.0, 1.0)
in

let similar_4 =
    if p_similar <= 0.05 then discrete(0.95, 0.05)
    else if p_similar <= 0.1 then discrete(0.9, 0.1)
    else if p_similar <= 0.15 then discrete(0.85, 0.15)
    else if p_similar <= 0.2 then discrete(0.8, 0.2)
    else if p_similar <= 0.25 then discrete(0.75, 0.25)
    else if p_similar <= 0.3 then discrete(0.7, 0.3)
    else if p_similar <= 0.35 then discrete(0.65, 0.35)
    else if p_similar <= 0.4 then discrete(0.6, 0.4)
    else if p_similar <= 0.45 then discrete(0.55, 0.45)
    else if p_similar <= 0.5 then discrete(0.5, 0.5)
    else if p_similar <= 0.55 then discrete(0.45, 0.55)
    else if p_similar <= 0.6 then discrete(0.4, 0.6)
    else if p_similar <= 0.65 then discrete(0.35, 0.65)
    else if p_similar <= 0.7 then discrete(0.3, 0.7)
    else if p_similar <= 0.75 then discrete(0.25, 0.75)
    else if p_similar <= 0.8 then discrete(0.2, 0.8)
    else if p_similar <= 0.85 then discrete(0.15, 0.85)
    else if p_similar <= 0.9 then discrete(0.1, 0.9)
    else if p_similar <= 0.95 then discrete(0.05, 0.95)
    else discrete(0.0, 1.0)
in

let p_click_url_A_4 =
    if similar_4 ==#2 1#2 then uniform(0,1) else uniform(0, 1)
in

let p_click_url_B_4 =
    if similar_4 ==#2 1#2 then p_click_url_A_4 else uniform(0, 1)
in

let click_url_A_4 =
    if p_click_url_A_4 <= 0.1 then discrete(0.9, 0.1)
    else if p_click_url_A_4 <= 0.2 then discrete(0.8, 0.2)
    else if p_click_url_A_4 <= 0.3 then discrete(0.7, 0.3)
    else if p_click_url_A_4 <= 0.4 then discrete(0.6, 0.4)
    else if p_click_url_A_4 <= 0.5 then discrete(0.5, 0.5)
    else if p_click_url_A_4 <= 0.6 then discrete(0.4, 0.6)
    else if p_click_url_A_4 <= 0.7 then discrete(0.3, 0.7)
    else if p_click_url_A_4 <= 0.8 then discrete(0.2, 0.8)
    else if p_click_url_A_4 <= 0.9 then discrete(0.1, 0.9)
    else discrete(0.0, 1.0)
in

let click_url_B_4 =
    if p_click_url_B_4 <= 0.1 then discrete(0.9, 0.1)
    else if p_click_url_B_4 <= 0.2 then discrete(0.8, 0.2)
    else if p_click_url_B_4 <= 0.3 then discrete(0.7, 0.3)
    else if p_click_url_B_4 <= 0.4 then discrete(0.6, 0.4)
```

```
    else if p_click_url_B_4 <= 0.5 then discrete(0.5, 0.5)
    else if p_click_url_B_4 <= 0.6 then discrete(0.4, 0.6)
    else if p_click_url_B_4 <= 0.7 then discrete(0.3, 0.7)
    else if p_click_url_B_4 <= 0.8 then discrete(0.2, 0.8)
    else if p_click_url_B_4 <= 0.9 then discrete(0.1, 0.9)
    else discrete(0.0, 1.0)
in

let _0 = observe(click_url_A_0 ==#2 1#2) in
let _1 = observe(click_url_B_0 ==#2 1#2) in
let _2 = observe(click_url_A_1 ==#2 1#2) in
let _3 = observe(click_url_B_1 ==#2 1#2) in
let _4 = observe(click_url_A_2 ==#2 0#2) in
let _5 = observe(click_url_B_2 ==#2 0#2) in
let _6 = observe(click_url_A_3 ==#2 1#2) in
let _7 = observe(click_url_B_3 ==#2 1#2) in
let _8 = observe(click_url_A_4 ==#2 0#2) in
let _9 = observe(click_url_B_4 ==#2 0#2) in

p_similar < 0.3
```

ClinicalTrial

```
let isEffective = discrete(0.5, 0.5) in
let probControl = discrete(0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05,
      0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05) in
let probTreated = discrete(0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05,
      0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05) in
let probAll = discrete(0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05,
      0.05, 0.05, 0.05, 0.05, 0.05, 0.05) in

let controlGroup0 =
    if isEffective ==#2 1#2 then
        if probControl ==#20 0#20 then discrete(1.0, 0.0)
        else if probControl ==#20 1#20 then discrete(0.95, 0.05)
        else if probControl ==#20 2#20 then discrete(0.9, 0.1)
        else if probControl ==#20 3#20 then discrete(0.85, 0.15)
        else if probControl ==#20 4#20 then discrete(0.8, 0.2)
        else if probControl ==#20 5#20 then discrete(0.75, 0.25)
        else if probControl ==#20 6#20 then discrete(0.7, 0.3)
        else if probControl ==#20 7#20 then discrete(0.65, 0.35)
        else if probControl ==#20 8#20 then discrete(0.6, 0.4)
        else if probControl ==#20 9#20 then discrete(0.55, 0.45)
        else if probControl ==#20 10#20 then discrete(0.5, 0.5)
        else if probControl ==#20 11#20 then discrete(0.45, 0.55)
        else if probControl ==#20 12#20 then discrete(0.4, 0.6)
        else if probControl ==#20 13#20 then discrete(0.35, 0.65)
        else if probControl ==#20 14#20 then discrete(0.3, 0.7)
        else if probControl ==#20 15#20 then discrete(0.25, 0.75)
        else if probControl ==#20 16#20 then discrete(0.2, 0.8)
        else if probControl ==#20 17#20 then discrete(0.15, 0.85)
        else if probControl ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95)
    else
        if probAll ==#20 0#20 then discrete(1.0, 0.0)
        else if probAll ==#20 1#20 then discrete(0.95, 0.05)
        else if probAll ==#20 2#20 then discrete(0.9, 0.1)
        else if probAll ==#20 3#20 then discrete(0.85, 0.15)
        else if probAll ==#20 4#20 then discrete(0.8, 0.2)
        else if probAll ==#20 5#20 then discrete(0.75, 0.25)
        else if probAll ==#20 6#20 then discrete(0.7, 0.3)
        else if probAll ==#20 7#20 then discrete(0.65, 0.35)
        else if probAll ==#20 8#20 then discrete(0.6, 0.4)
        else if probAll ==#20 9#20 then discrete(0.55, 0.45)
        else if probAll ==#20 10#20 then discrete(0.5, 0.5)
        else if probAll ==#20 11#20 then discrete(0.45, 0.55)
        else if probAll ==#20 12#20 then discrete(0.4, 0.6)
        else if probAll ==#20 13#20 then discrete(0.35, 0.65)
        else if probAll ==#20 14#20 then discrete(0.3, 0.7)
        else if probAll ==#20 15#20 then discrete(0.25, 0.75)
```

```
        else if probAll ==#20 16#20 then discrete(0.2, 0.8)
        else if probAll ==#20 17#20 then discrete(0.15, 0.85)
        else if probAll ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95) in

let treatedGroup0 =
    if isEffective ==#2 1#2 then
        if probTreated ==#20 0#20 then discrete(1.0, 0.0)
        else if probTreated ==#20 1#20 then discrete(0.95, 0.05)
        else if probTreated ==#20 2#20 then discrete(0.9, 0.1)
        else if probTreated ==#20 3#20 then discrete(0.85, 0.15)
        else if probTreated ==#20 4#20 then discrete(0.8, 0.2)
        else if probTreated ==#20 5#20 then discrete(0.75, 0.25)
        else if probTreated ==#20 6#20 then discrete(0.7, 0.3)
        else if probTreated ==#20 7#20 then discrete(0.65, 0.35)
        else if probTreated ==#20 8#20 then discrete(0.6, 0.4)
        else if probTreated ==#20 9#20 then discrete(0.55, 0.45)
        else if probTreated ==#20 10#20 then discrete(0.5, 0.5)
        else if probTreated ==#20 11#20 then discrete(0.45, 0.55)
        else if probTreated ==#20 12#20 then discrete(0.4, 0.6)
        else if probTreated ==#20 13#20 then discrete(0.35, 0.65)
        else if probTreated ==#20 14#20 then discrete(0.3, 0.7)
        else if probTreated ==#20 15#20 then discrete(0.25, 0.75)
        else if probTreated ==#20 16#20 then discrete(0.2, 0.8)
        else if probTreated ==#20 17#20 then discrete(0.15, 0.85)
        else if probTreated ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95)
    else
        if probAll ==#20 0#20 then discrete(1.0, 0.0)
        else if probAll ==#20 1#20 then discrete(0.95, 0.05)
        else if probAll ==#20 2#20 then discrete(0.9, 0.1)
        else if probAll ==#20 3#20 then discrete(0.85, 0.15)
        else if probAll ==#20 4#20 then discrete(0.8, 0.2)
        else if probAll ==#20 5#20 then discrete(0.75, 0.25)
        else if probAll ==#20 6#20 then discrete(0.7, 0.3)
        else if probAll ==#20 7#20 then discrete(0.65, 0.35)
        else if probAll ==#20 8#20 then discrete(0.6, 0.4)
        else if probAll ==#20 9#20 then discrete(0.55, 0.45)
        else if probAll ==#20 10#20 then discrete(0.5, 0.5)
        else if probAll ==#20 11#20 then discrete(0.45, 0.55)
        else if probAll ==#20 12#20 then discrete(0.4, 0.6)
        else if probAll ==#20 13#20 then discrete(0.35, 0.65)
        else if probAll ==#20 14#20 then discrete(0.3, 0.7)
        else if probAll ==#20 15#20 then discrete(0.25, 0.75)
        else if probAll ==#20 16#20 then discrete(0.2, 0.8)
        else if probAll ==#20 17#20 then discrete(0.15, 0.85)
        else if probAll ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95) in

let controlGroup1 =
    if isEffective ==#2 1#2 then
        if probControl ==#20 0#20 then discrete(1.0, 0.0)
        else if probControl ==#20 1#20 then discrete(0.95, 0.05)
        else if probControl ==#20 2#20 then discrete(0.9, 0.1)
        else if probControl ==#20 3#20 then discrete(0.85, 0.15)
        else if probControl ==#20 4#20 then discrete(0.8, 0.2)
        else if probControl ==#20 5#20 then discrete(0.75, 0.25)
        else if probControl ==#20 6#20 then discrete(0.7, 0.3)
        else if probControl ==#20 7#20 then discrete(0.65, 0.35)
        else if probControl ==#20 8#20 then discrete(0.6, 0.4)
        else if probControl ==#20 9#20 then discrete(0.55, 0.45)
        else if probControl ==#20 10#20 then discrete(0.5, 0.5)
        else if probControl ==#20 11#20 then discrete(0.45, 0.55)
        else if probControl ==#20 12#20 then discrete(0.4, 0.6)
        else if probControl ==#20 13#20 then discrete(0.35, 0.65)
        else if probControl ==#20 14#20 then discrete(0.3, 0.7)
        else if probControl ==#20 15#20 then discrete(0.25, 0.75)
        else if probControl ==#20 16#20 then discrete(0.2, 0.8)
        else if probControl ==#20 17#20 then discrete(0.15, 0.85)
        else if probControl ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95)
```

```
    else
        if probAll ==#20 0#20 then discrete(1.0, 0.0)
        else if probAll ==#20 1#20 then discrete(0.95, 0.05)
        else if probAll ==#20 2#20 then discrete(0.9, 0.1)
        else if probAll ==#20 3#20 then discrete(0.85, 0.15)
        else if probAll ==#20 4#20 then discrete(0.8, 0.2)
        else if probAll ==#20 5#20 then discrete(0.75, 0.25)
        else if probAll ==#20 6#20 then discrete(0.7, 0.3)
        else if probAll ==#20 7#20 then discrete(0.65, 0.35)
        else if probAll ==#20 8#20 then discrete(0.6, 0.4)
        else if probAll ==#20 9#20 then discrete(0.55, 0.45)
        else if probAll ==#20 10#20 then discrete(0.5, 0.5)
        else if probAll ==#20 11#20 then discrete(0.45, 0.55)
        else if probAll ==#20 12#20 then discrete(0.4, 0.6)
        else if probAll ==#20 13#20 then discrete(0.35, 0.65)
        else if probAll ==#20 14#20 then discrete(0.3, 0.7)
        else if probAll ==#20 15#20 then discrete(0.25, 0.75)
        else if probAll ==#20 16#20 then discrete(0.2, 0.8)
        else if probAll ==#20 17#20 then discrete(0.15, 0.85)
        else if probAll ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95) in

let treatedGroup1 =
    if isEffective ==#2 1#2 then
        if probTreated ==#20 0#20 then discrete(1.0, 0.0)
        else if probTreated ==#20 1#20 then discrete(0.95, 0.05)
        else if probTreated ==#20 2#20 then discrete(0.9, 0.1)
        else if probTreated ==#20 3#20 then discrete(0.85, 0.15)
        else if probTreated ==#20 4#20 then discrete(0.8, 0.2)
        else if probTreated ==#20 5#20 then discrete(0.75, 0.25)
        else if probTreated ==#20 6#20 then discrete(0.7, 0.3)
        else if probTreated ==#20 7#20 then discrete(0.65, 0.35)
        else if probTreated ==#20 8#20 then discrete(0.6, 0.4)
        else if probTreated ==#20 9#20 then discrete(0.55, 0.45)
        else if probTreated ==#20 10#20 then discrete(0.5, 0.5)
        else if probTreated ==#20 11#20 then discrete(0.45, 0.55)
        else if probTreated ==#20 12#20 then discrete(0.4, 0.6)
        else if probTreated ==#20 13#20 then discrete(0.35, 0.65)
        else if probTreated ==#20 14#20 then discrete(0.3, 0.7)
        else if probTreated ==#20 15#20 then discrete(0.25, 0.75)
        else if probTreated ==#20 16#20 then discrete(0.2, 0.8)
        else if probTreated ==#20 17#20 then discrete(0.15, 0.85)
        else if probTreated ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95)
    else
        if probAll ==#20 0#20 then discrete(1.0, 0.0)
        else if probAll ==#20 1#20 then discrete(0.95, 0.05)
        else if probAll ==#20 2#20 then discrete(0.9, 0.1)
        else if probAll ==#20 3#20 then discrete(0.85, 0.15)
        else if probAll ==#20 4#20 then discrete(0.8, 0.2)
        else if probAll ==#20 5#20 then discrete(0.75, 0.25)
        else if probAll ==#20 6#20 then discrete(0.7, 0.3)
        else if probAll ==#20 7#20 then discrete(0.65, 0.35)
        else if probAll ==#20 8#20 then discrete(0.6, 0.4)
        else if probAll ==#20 9#20 then discrete(0.55, 0.45)
        else if probAll ==#20 10#20 then discrete(0.5, 0.5)
        else if probAll ==#20 11#20 then discrete(0.45, 0.55)
        else if probAll ==#20 12#20 then discrete(0.4, 0.6)
        else if probAll ==#20 13#20 then discrete(0.35, 0.65)
        else if probAll ==#20 14#20 then discrete(0.3, 0.7)
        else if probAll ==#20 15#20 then discrete(0.25, 0.75)
        else if probAll ==#20 16#20 then discrete(0.2, 0.8)
        else if probAll ==#20 17#20 then discrete(0.15, 0.85)
        else if probAll ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95) in

let controlGroup2 =
    if isEffective ==#2 1#2 then
        if probControl ==#20 0#20 then discrete(1.0, 0.0)
        else if probControl ==#20 1#20 then discrete(0.95, 0.05)
        else if probControl ==#20 2#20 then discrete(0.9, 0.1)
```

```
        else if probControl ==#20 3#20 then discrete(0.85, 0.15)
        else if probControl ==#20 4#20 then discrete(0.8, 0.2)
        else if probControl ==#20 5#20 then discrete(0.75, 0.25)
        else if probControl ==#20 6#20 then discrete(0.7, 0.3)
        else if probControl ==#20 7#20 then discrete(0.65, 0.35)
        else if probControl ==#20 8#20 then discrete(0.6, 0.4)
        else if probControl ==#20 9#20 then discrete(0.55, 0.45)
        else if probControl ==#20 10#20 then discrete(0.5, 0.5)
        else if probControl ==#20 11#20 then discrete(0.45, 0.55)
        else if probControl ==#20 12#20 then discrete(0.4, 0.6)
        else if probControl ==#20 13#20 then discrete(0.35, 0.65)
        else if probControl ==#20 14#20 then discrete(0.3, 0.7)
        else if probControl ==#20 15#20 then discrete(0.25, 0.75)
        else if probControl ==#20 16#20 then discrete(0.2, 0.8)
        else if probControl ==#20 17#20 then discrete(0.15, 0.85)
        else if probControl ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95)
    else
        if probAll ==#20 0#20 then discrete(1.0, 0.0)
        else if probAll ==#20 1#20 then discrete(0.95, 0.05)
        else if probAll ==#20 2#20 then discrete(0.9, 0.1)
        else if probAll ==#20 3#20 then discrete(0.85, 0.15)
        else if probAll ==#20 4#20 then discrete(0.8, 0.2)
        else if probAll ==#20 5#20 then discrete(0.75, 0.25)
        else if probAll ==#20 6#20 then discrete(0.7, 0.3)
        else if probAll ==#20 7#20 then discrete(0.65, 0.35)
        else if probAll ==#20 8#20 then discrete(0.6, 0.4)
        else if probAll ==#20 9#20 then discrete(0.55, 0.45)
        else if probAll ==#20 10#20 then discrete(0.5, 0.5)
        else if probAll ==#20 11#20 then discrete(0.45, 0.55)
        else if probAll ==#20 12#20 then discrete(0.4, 0.6)
        else if probAll ==#20 13#20 then discrete(0.35, 0.65)
        else if probAll ==#20 14#20 then discrete(0.3, 0.7)
        else if probAll ==#20 15#20 then discrete(0.25, 0.75)
        else if probAll ==#20 16#20 then discrete(0.2, 0.8)
        else if probAll ==#20 17#20 then discrete(0.15, 0.85)
        else if probAll ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95) in

let treatedGroup2 =
    if isEffective ==#2 1#2 then
        if probTreated ==#20 0#20 then discrete(1.0, 0.0)
        else if probTreated ==#20 1#20 then discrete(0.95, 0.05)
        else if probTreated ==#20 2#20 then discrete(0.9, 0.1)
        else if probTreated ==#20 3#20 then discrete(0.85, 0.15)
        else if probTreated ==#20 4#20 then discrete(0.8, 0.2)
        else if probTreated ==#20 5#20 then discrete(0.75, 0.25)
        else if probTreated ==#20 6#20 then discrete(0.7, 0.3)
        else if probTreated ==#20 7#20 then discrete(0.65, 0.35)
        else if probTreated ==#20 8#20 then discrete(0.6, 0.4)
        else if probTreated ==#20 9#20 then discrete(0.55, 0.45)
        else if probTreated ==#20 10#20 then discrete(0.5, 0.5)
        else if probTreated ==#20 11#20 then discrete(0.45, 0.55)
        else if probTreated ==#20 12#20 then discrete(0.4, 0.6)
        else if probTreated ==#20 13#20 then discrete(0.35, 0.65)
        else if probTreated ==#20 14#20 then discrete(0.3, 0.7)
        else if probTreated ==#20 15#20 then discrete(0.25, 0.75)
        else if probTreated ==#20 16#20 then discrete(0.2, 0.8)
        else if probTreated ==#20 17#20 then discrete(0.15, 0.85)
        else if probTreated ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95)
    else
        if probAll ==#20 0#20 then discrete(1.0, 0.0)
        else if probAll ==#20 1#20 then discrete(0.95, 0.05)
        else if probAll ==#20 2#20 then discrete(0.9, 0.1)
        else if probAll ==#20 3#20 then discrete(0.85, 0.15)
        else if probAll ==#20 4#20 then discrete(0.8, 0.2)
        else if probAll ==#20 5#20 then discrete(0.75, 0.25)
        else if probAll ==#20 6#20 then discrete(0.7, 0.3)
        else if probAll ==#20 7#20 then discrete(0.65, 0.35)
        else if probAll ==#20 8#20 then discrete(0.6, 0.4)
```

```
        else if probAll ==#20 9#20 then discrete(0.55, 0.45)
        else if probAll ==#20 10#20 then discrete(0.5, 0.5)
        else if probAll ==#20 11#20 then discrete(0.45, 0.55)
        else if probAll ==#20 12#20 then discrete(0.4, 0.6)
        else if probAll ==#20 13#20 then discrete(0.35, 0.65)
        else if probAll ==#20 14#20 then discrete(0.3, 0.7)
        else if probAll ==#20 15#20 then discrete(0.25, 0.75)
        else if probAll ==#20 16#20 then discrete(0.2, 0.8)
        else if probAll ==#20 17#20 then discrete(0.15, 0.85)
        else if probAll ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95) in

let controlGroup3 =
    if isEffective ==#2 1#2 then
        if probControl ==#20 0#20 then discrete(1.0, 0.0)
        else if probControl ==#20 1#20 then discrete(0.95, 0.05)
        else if probControl ==#20 2#20 then discrete(0.9, 0.1)
        else if probControl ==#20 3#20 then discrete(0.85, 0.15)
        else if probControl ==#20 4#20 then discrete(0.8, 0.2)
        else if probControl ==#20 5#20 then discrete(0.75, 0.25)
        else if probControl ==#20 6#20 then discrete(0.7, 0.3)
        else if probControl ==#20 7#20 then discrete(0.65, 0.35)
        else if probControl ==#20 8#20 then discrete(0.6, 0.4)
        else if probControl ==#20 9#20 then discrete(0.55, 0.45)
        else if probControl ==#20 10#20 then discrete(0.5, 0.5)
        else if probControl ==#20 11#20 then discrete(0.45, 0.55)
        else if probControl ==#20 12#20 then discrete(0.4, 0.6)
        else if probControl ==#20 13#20 then discrete(0.35, 0.65)
        else if probControl ==#20 14#20 then discrete(0.3, 0.7)
        else if probControl ==#20 15#20 then discrete(0.25, 0.75)
        else if probControl ==#20 16#20 then discrete(0.2, 0.8)
        else if probControl ==#20 17#20 then discrete(0.15, 0.85)
        else if probControl ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95)
    else
        if probAll ==#20 0#20 then discrete(1.0, 0.0)
        else if probAll ==#20 1#20 then discrete(0.95, 0.05)
        else if probAll ==#20 2#20 then discrete(0.9, 0.1)
        else if probAll ==#20 3#20 then discrete(0.85, 0.15)
        else if probAll ==#20 4#20 then discrete(0.8, 0.2)
        else if probAll ==#20 5#20 then discrete(0.75, 0.25)
        else if probAll ==#20 6#20 then discrete(0.7, 0.3)
        else if probAll ==#20 7#20 then discrete(0.65, 0.35)
        else if probAll ==#20 8#20 then discrete(0.6, 0.4)
        else if probAll ==#20 9#20 then discrete(0.55, 0.45)
        else if probAll ==#20 10#20 then discrete(0.5, 0.5)
        else if probAll ==#20 11#20 then discrete(0.45, 0.55)
        else if probAll ==#20 12#20 then discrete(0.4, 0.6)
        else if probAll ==#20 13#20 then discrete(0.35, 0.65)
        else if probAll ==#20 14#20 then discrete(0.3, 0.7)
        else if probAll ==#20 15#20 then discrete(0.25, 0.75)
        else if probAll ==#20 16#20 then discrete(0.2, 0.8)
        else if probAll ==#20 17#20 then discrete(0.15, 0.85)
        else if probAll ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95) in

let treatedGroup3 =
    if isEffective ==#2 1#2 then
        if probTreated ==#20 0#20 then discrete(1.0, 0.0)
        else if probTreated ==#20 1#20 then discrete(0.95, 0.05)
        else if probTreated ==#20 2#20 then discrete(0.9, 0.1)
        else if probTreated ==#20 3#20 then discrete(0.85, 0.15)
        else if probTreated ==#20 4#20 then discrete(0.8, 0.2)
        else if probTreated ==#20 5#20 then discrete(0.75, 0.25)
        else if probTreated ==#20 6#20 then discrete(0.7, 0.3)
        else if probTreated ==#20 7#20 then discrete(0.65, 0.35)
        else if probTreated ==#20 8#20 then discrete(0.6, 0.4)
        else if probTreated ==#20 9#20 then discrete(0.55, 0.45)
        else if probTreated ==#20 10#20 then discrete(0.5, 0.5)
        else if probTreated ==#20 11#20 then discrete(0.45, 0.55)
        else if probTreated ==#20 12#20 then discrete(0.4, 0.6)
```

```
        else if probTreated ==#20 13#20 then discrete(0.35, 0.65)
        else if probTreated ==#20 14#20 then discrete(0.3, 0.7)
        else if probTreated ==#20 15#20 then discrete(0.25, 0.75)
        else if probTreated ==#20 16#20 then discrete(0.2, 0.8)
        else if probTreated ==#20 17#20 then discrete(0.15, 0.85)
        else if probTreated ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95)
    else
        if probAll ==#20 0#20 then discrete(1.0, 0.0)
        else if probAll ==#20 1#20 then discrete(0.95, 0.05)
        else if probAll ==#20 2#20 then discrete(0.9, 0.1)
        else if probAll ==#20 3#20 then discrete(0.85, 0.15)
        else if probAll ==#20 4#20 then discrete(0.8, 0.2)
        else if probAll ==#20 5#20 then discrete(0.75, 0.25)
        else if probAll ==#20 6#20 then discrete(0.7, 0.3)
        else if probAll ==#20 7#20 then discrete(0.65, 0.35)
        else if probAll ==#20 8#20 then discrete(0.6, 0.4)
        else if probAll ==#20 9#20 then discrete(0.55, 0.45)
        else if probAll ==#20 10#20 then discrete(0.5, 0.5)
        else if probAll ==#20 11#20 then discrete(0.45, 0.55)
        else if probAll ==#20 12#20 then discrete(0.4, 0.6)
        else if probAll ==#20 13#20 then discrete(0.35, 0.65)
        else if probAll ==#20 14#20 then discrete(0.3, 0.7)
        else if probAll ==#20 15#20 then discrete(0.25, 0.75)
        else if probAll ==#20 16#20 then discrete(0.2, 0.8)
        else if probAll ==#20 17#20 then discrete(0.15, 0.85)
        else if probAll ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95) in

let controlGroup4 =
    if isEffective ==#2 1#2 then
        if probControl ==#20 0#20 then discrete(1.0, 0.0)
        else if probControl ==#20 1#20 then discrete(0.95, 0.05)
        else if probControl ==#20 2#20 then discrete(0.9, 0.1)
        else if probControl ==#20 3#20 then discrete(0.85, 0.15)
        else if probControl ==#20 4#20 then discrete(0.8, 0.2)
        else if probControl ==#20 5#20 then discrete(0.75, 0.25)
        else if probControl ==#20 6#20 then discrete(0.7, 0.3)
        else if probControl ==#20 7#20 then discrete(0.65, 0.35)
        else if probControl ==#20 8#20 then discrete(0.6, 0.4)
        else if probControl ==#20 9#20 then discrete(0.55, 0.45)
        else if probControl ==#20 10#20 then discrete(0.5, 0.5)
        else if probControl ==#20 11#20 then discrete(0.45, 0.55)
        else if probControl ==#20 12#20 then discrete(0.4, 0.6)
        else if probControl ==#20 13#20 then discrete(0.35, 0.65)
        else if probControl ==#20 14#20 then discrete(0.3, 0.7)
        else if probControl ==#20 15#20 then discrete(0.25, 0.75)
        else if probControl ==#20 16#20 then discrete(0.2, 0.8)
        else if probControl ==#20 17#20 then discrete(0.15, 0.85)
        else if probControl ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95)
    else
        if probAll ==#20 0#20 then discrete(1.0, 0.0)
        else if probAll ==#20 1#20 then discrete(0.95, 0.05)
        else if probAll ==#20 2#20 then discrete(0.9, 0.1)
        else if probAll ==#20 3#20 then discrete(0.85, 0.15)
        else if probAll ==#20 4#20 then discrete(0.8, 0.2)
        else if probAll ==#20 5#20 then discrete(0.75, 0.25)
        else if probAll ==#20 6#20 then discrete(0.7, 0.3)
        else if probAll ==#20 7#20 then discrete(0.65, 0.35)
        else if probAll ==#20 8#20 then discrete(0.6, 0.4)
        else if probAll ==#20 9#20 then discrete(0.55, 0.45)
        else if probAll ==#20 10#20 then discrete(0.5, 0.5)
        else if probAll ==#20 11#20 then discrete(0.45, 0.55)
        else if probAll ==#20 12#20 then discrete(0.4, 0.6)
        else if probAll ==#20 13#20 then discrete(0.35, 0.65)
        else if probAll ==#20 14#20 then discrete(0.3, 0.7)
        else if probAll ==#20 15#20 then discrete(0.25, 0.75)
        else if probAll ==#20 16#20 then discrete(0.2, 0.8)
        else if probAll ==#20 17#20 then discrete(0.15, 0.85)
        else if probAll ==#20 18#20 then discrete(0.1, 0.9)
```

```
        else discrete(0.05, 0.95) in

let treatedGroup4 =
    if isEffective ==#2 1#2 then
        if probTreated ==#20 0#20 then discrete(1.0, 0.0)
        else if probTreated ==#20 1#20 then discrete(0.95, 0.05)
        else if probTreated ==#20 2#20 then discrete(0.9, 0.1)
        else if probTreated ==#20 3#20 then discrete(0.85, 0.15)
        else if probTreated ==#20 4#20 then discrete(0.8, 0.2)
        else if probTreated ==#20 5#20 then discrete(0.75, 0.25)
        else if probTreated ==#20 6#20 then discrete(0.7, 0.3)
        else if probTreated ==#20 7#20 then discrete(0.65, 0.35)
        else if probTreated ==#20 8#20 then discrete(0.6, 0.4)
        else if probTreated ==#20 9#20 then discrete(0.55, 0.45)
        else if probTreated ==#20 10#20 then discrete(0.5, 0.5)
        else if probTreated ==#20 11#20 then discrete(0.45, 0.55)
        else if probTreated ==#20 12#20 then discrete(0.4, 0.6)
        else if probTreated ==#20 13#20 then discrete(0.35, 0.65)
        else if probTreated ==#20 14#20 then discrete(0.3, 0.7)
        else if probTreated ==#20 15#20 then discrete(0.25, 0.75)
        else if probTreated ==#20 16#20 then discrete(0.2, 0.8)
        else if probTreated ==#20 17#20 then discrete(0.15, 0.85)
        else if probTreated ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95)
    else
        if probAll ==#20 0#20 then discrete(1.0, 0.0)
        else if probAll ==#20 1#20 then discrete(0.95, 0.05)
        else if probAll ==#20 2#20 then discrete(0.9, 0.1)
        else if probAll ==#20 3#20 then discrete(0.85, 0.15)
        else if probAll ==#20 4#20 then discrete(0.8, 0.2)
        else if probAll ==#20 5#20 then discrete(0.75, 0.25)
        else if probAll ==#20 6#20 then discrete(0.7, 0.3)
        else if probAll ==#20 7#20 then discrete(0.65, 0.35)
        else if probAll ==#20 8#20 then discrete(0.6, 0.4)
        else if probAll ==#20 9#20 then discrete(0.55, 0.45)
        else if probAll ==#20 10#20 then discrete(0.5, 0.5)
        else if probAll ==#20 11#20 then discrete(0.45, 0.55)
        else if probAll ==#20 12#20 then discrete(0.4, 0.6)
        else if probAll ==#20 13#20 then discrete(0.35, 0.65)
        else if probAll ==#20 14#20 then discrete(0.3, 0.7)
        else if probAll ==#20 15#20 then discrete(0.25, 0.75)
        else if probAll ==#20 16#20 then discrete(0.2, 0.8)
        else if probAll ==#20 17#20 then discrete(0.15, 0.85)
        else if probAll ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95) in

let controlGroup5 =
    if isEffective ==#2 1#2 then
        if probControl ==#20 0#20 then discrete(1.0, 0.0)
        else if probControl ==#20 1#20 then discrete(0.95, 0.05)
        else if probControl ==#20 2#20 then discrete(0.9, 0.1)
        else if probControl ==#20 3#20 then discrete(0.85, 0.15)
        else if probControl ==#20 4#20 then discrete(0.8, 0.2)
        else if probControl ==#20 5#20 then discrete(0.75, 0.25)
        else if probControl ==#20 6#20 then discrete(0.7, 0.3)
        else if probControl ==#20 7#20 then discrete(0.65, 0.35)
        else if probControl ==#20 8#20 then discrete(0.6, 0.4)
        else if probControl ==#20 9#20 then discrete(0.55, 0.45)
        else if probControl ==#20 10#20 then discrete(0.5, 0.5)
        else if probControl ==#20 11#20 then discrete(0.45, 0.55)
        else if probControl ==#20 12#20 then discrete(0.4, 0.6)
        else if probControl ==#20 13#20 then discrete(0.35, 0.65)
        else if probControl ==#20 14#20 then discrete(0.3, 0.7)
        else if probControl ==#20 15#20 then discrete(0.25, 0.75)
        else if probControl ==#20 16#20 then discrete(0.2, 0.8)
        else if probControl ==#20 17#20 then discrete(0.15, 0.85)
        else if probControl ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95)
    else
        if probAll ==#20 0#20 then discrete(1.0, 0.0)
        else if probAll ==#20 1#20 then discrete(0.95, 0.05)
```

```
        else if probAll ==#20 2#20 then discrete(0.9, 0.1)
        else if probAll ==#20 3#20 then discrete(0.85, 0.15)
        else if probAll ==#20 4#20 then discrete(0.8, 0.2)
        else if probAll ==#20 5#20 then discrete(0.75, 0.25)
        else if probAll ==#20 6#20 then discrete(0.7, 0.3)
        else if probAll ==#20 7#20 then discrete(0.65, 0.35)
        else if probAll ==#20 8#20 then discrete(0.6, 0.4)
        else if probAll ==#20 9#20 then discrete(0.55, 0.45)
        else if probAll ==#20 10#20 then discrete(0.5, 0.5)
        else if probAll ==#20 11#20 then discrete(0.45, 0.55)
        else if probAll ==#20 12#20 then discrete(0.4, 0.6)
        else if probAll ==#20 13#20 then discrete(0.35, 0.65)
        else if probAll ==#20 14#20 then discrete(0.3, 0.7)
        else if probAll ==#20 15#20 then discrete(0.25, 0.75)
        else if probAll ==#20 16#20 then discrete(0.2, 0.8)
        else if probAll ==#20 17#20 then discrete(0.15, 0.85)
        else if probAll ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95) in

let treatedGroup5 =
    if isEffective ==#2 1#2 then
        if probTreated ==#20 0#20 then discrete(1.0, 0.0)
        else if probTreated ==#20 1#20 then discrete(0.95, 0.05)
        else if probTreated ==#20 2#20 then discrete(0.9, 0.1)
        else if probTreated ==#20 3#20 then discrete(0.85, 0.15)
        else if probTreated ==#20 4#20 then discrete(0.8, 0.2)
        else if probTreated ==#20 5#20 then discrete(0.75, 0.25)
        else if probTreated ==#20 6#20 then discrete(0.7, 0.3)
        else if probTreated ==#20 7#20 then discrete(0.65, 0.35)
        else if probTreated ==#20 8#20 then discrete(0.6, 0.4)
        else if probTreated ==#20 9#20 then discrete(0.55, 0.45)
        else if probTreated ==#20 10#20 then discrete(0.5, 0.5)
        else if probTreated ==#20 11#20 then discrete(0.45, 0.55)
        else if probTreated ==#20 12#20 then discrete(0.4, 0.6)
        else if probTreated ==#20 13#20 then discrete(0.35, 0.65)
        else if probTreated ==#20 14#20 then discrete(0.3, 0.7)
        else if probTreated ==#20 15#20 then discrete(0.25, 0.75)
        else if probTreated ==#20 16#20 then discrete(0.2, 0.8)
        else if probTreated ==#20 17#20 then discrete(0.15, 0.85)
        else if probTreated ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95)
    else
        if probAll ==#20 0#20 then discrete(1.0, 0.0)
        else if probAll ==#20 1#20 then discrete(0.95, 0.05)
        else if probAll ==#20 2#20 then discrete(0.9, 0.1)
        else if probAll ==#20 3#20 then discrete(0.85, 0.15)
        else if probAll ==#20 4#20 then discrete(0.8, 0.2)
        else if probAll ==#20 5#20 then discrete(0.75, 0.25)
        else if probAll ==#20 6#20 then discrete(0.7, 0.3)
        else if probAll ==#20 7#20 then discrete(0.65, 0.35)
        else if probAll ==#20 8#20 then discrete(0.6, 0.4)
        else if probAll ==#20 9#20 then discrete(0.55, 0.45)
        else if probAll ==#20 10#20 then discrete(0.5, 0.5)
        else if probAll ==#20 11#20 then discrete(0.45, 0.55)
        else if probAll ==#20 12#20 then discrete(0.4, 0.6)
        else if probAll ==#20 13#20 then discrete(0.35, 0.65)
        else if probAll ==#20 14#20 then discrete(0.3, 0.7)
        else if probAll ==#20 15#20 then discrete(0.25, 0.75)
        else if probAll ==#20 16#20 then discrete(0.2, 0.8)
        else if probAll ==#20 17#20 then discrete(0.15, 0.85)
        else if probAll ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95) in

let controlGroup6 =
    if isEffective ==#2 1#2 then
        if probControl ==#20 0#20 then discrete(1.0, 0.0)
        else if probControl ==#20 1#20 then discrete(0.95, 0.05)
        else if probControl ==#20 2#20 then discrete(0.9, 0.1)
        else if probControl ==#20 3#20 then discrete(0.85, 0.15)
        else if probControl ==#20 4#20 then discrete(0.8, 0.2)
        else if probControl ==#20 5#20 then discrete(0.75, 0.25)
```

```
        else if probControl ==#20 6#20 then discrete(0.7, 0.3)
        else if probControl ==#20 7#20 then discrete(0.65, 0.35)
        else if probControl ==#20 8#20 then discrete(0.6, 0.4)
        else if probControl ==#20 9#20 then discrete(0.55, 0.45)
        else if probControl ==#20 10#20 then discrete(0.5, 0.5)
        else if probControl ==#20 11#20 then discrete(0.45, 0.55)
        else if probControl ==#20 12#20 then discrete(0.4, 0.6)
        else if probControl ==#20 13#20 then discrete(0.35, 0.65)
        else if probControl ==#20 14#20 then discrete(0.3, 0.7)
        else if probControl ==#20 15#20 then discrete(0.25, 0.75)
        else if probControl ==#20 16#20 then discrete(0.2, 0.8)
        else if probControl ==#20 17#20 then discrete(0.15, 0.85)
        else if probControl ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95)
    else
        if probAll ==#20 0#20 then discrete(1.0, 0.0)
        else if probAll ==#20 1#20 then discrete(0.95, 0.05)
        else if probAll ==#20 2#20 then discrete(0.9, 0.1)
        else if probAll ==#20 3#20 then discrete(0.85, 0.15)
        else if probAll ==#20 4#20 then discrete(0.8, 0.2)
        else if probAll ==#20 5#20 then discrete(0.75, 0.25)
        else if probAll ==#20 6#20 then discrete(0.7, 0.3)
        else if probAll ==#20 7#20 then discrete(0.65, 0.35)
        else if probAll ==#20 8#20 then discrete(0.6, 0.4)
        else if probAll ==#20 9#20 then discrete(0.55, 0.45)
        else if probAll ==#20 10#20 then discrete(0.5, 0.5)
        else if probAll ==#20 11#20 then discrete(0.45, 0.55)
        else if probAll ==#20 12#20 then discrete(0.4, 0.6)
        else if probAll ==#20 13#20 then discrete(0.35, 0.65)
        else if probAll ==#20 14#20 then discrete(0.3, 0.7)
        else if probAll ==#20 15#20 then discrete(0.25, 0.75)
        else if probAll ==#20 16#20 then discrete(0.2, 0.8)
        else if probAll ==#20 17#20 then discrete(0.15, 0.85)
        else if probAll ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95) in

let treatedGroup6 =
    if isEffective ==#2 1#2 then
        if probTreated ==#20 0#20 then discrete(1.0, 0.0)
        else if probTreated ==#20 1#20 then discrete(0.95, 0.05)
        else if probTreated ==#20 2#20 then discrete(0.9, 0.1)
        else if probTreated ==#20 3#20 then discrete(0.85, 0.15)
        else if probTreated ==#20 4#20 then discrete(0.8, 0.2)
        else if probTreated ==#20 5#20 then discrete(0.75, 0.25)
        else if probTreated ==#20 6#20 then discrete(0.7, 0.3)
        else if probTreated ==#20 7#20 then discrete(0.65, 0.35)
        else if probTreated ==#20 8#20 then discrete(0.6, 0.4)
        else if probTreated ==#20 9#20 then discrete(0.55, 0.45)
        else if probTreated ==#20 10#20 then discrete(0.5, 0.5)
        else if probTreated ==#20 11#20 then discrete(0.45, 0.55)
        else if probTreated ==#20 12#20 then discrete(0.4, 0.6)
        else if probTreated ==#20 13#20 then discrete(0.35, 0.65)
        else if probTreated ==#20 14#20 then discrete(0.3, 0.7)
        else if probTreated ==#20 15#20 then discrete(0.25, 0.75)
        else if probTreated ==#20 16#20 then discrete(0.2, 0.8)
        else if probTreated ==#20 17#20 then discrete(0.15, 0.85)
        else if probTreated ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95)
    else
        if probAll ==#20 0#20 then discrete(1.0, 0.0)
        else if probAll ==#20 1#20 then discrete(0.95, 0.05)
        else if probAll ==#20 2#20 then discrete(0.9, 0.1)
        else if probAll ==#20 3#20 then discrete(0.85, 0.15)
        else if probAll ==#20 4#20 then discrete(0.8, 0.2)
        else if probAll ==#20 5#20 then discrete(0.75, 0.25)
        else if probAll ==#20 6#20 then discrete(0.7, 0.3)
        else if probAll ==#20 7#20 then discrete(0.65, 0.35)
        else if probAll ==#20 8#20 then discrete(0.6, 0.4)
        else if probAll ==#20 9#20 then discrete(0.55, 0.45)
        else if probAll ==#20 10#20 then discrete(0.5, 0.5)
        else if probAll ==#20 11#20 then discrete(0.45, 0.55)
```

```
        else if probAll ==#20 12#20 then discrete(0.4, 0.6)
        else if probAll ==#20 13#20 then discrete(0.35, 0.65)
        else if probAll ==#20 14#20 then discrete(0.3, 0.7)
        else if probAll ==#20 15#20 then discrete(0.25, 0.75)
        else if probAll ==#20 16#20 then discrete(0.2, 0.8)
        else if probAll ==#20 17#20 then discrete(0.15, 0.85)
        else if probAll ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95) in

let controlGroup7 =
    if isEffective ==#2 1#2 then
        if probControl ==#20 0#20 then discrete(1.0, 0.0)
        else if probControl ==#20 1#20 then discrete(0.95, 0.05)
        else if probControl ==#20 2#20 then discrete(0.9, 0.1)
        else if probControl ==#20 3#20 then discrete(0.85, 0.15)
        else if probControl ==#20 4#20 then discrete(0.8, 0.2)
        else if probControl ==#20 5#20 then discrete(0.75, 0.25)
        else if probControl ==#20 6#20 then discrete(0.7, 0.3)
        else if probControl ==#20 7#20 then discrete(0.65, 0.35)
        else if probControl ==#20 8#20 then discrete(0.6, 0.4)
        else if probControl ==#20 9#20 then discrete(0.55, 0.45)
        else if probControl ==#20 10#20 then discrete(0.5, 0.5)
        else if probControl ==#20 11#20 then discrete(0.45, 0.55)
        else if probControl ==#20 12#20 then discrete(0.4, 0.6)
        else if probControl ==#20 13#20 then discrete(0.35, 0.65)
        else if probControl ==#20 14#20 then discrete(0.3, 0.7)
        else if probControl ==#20 15#20 then discrete(0.25, 0.75)
        else if probControl ==#20 16#20 then discrete(0.2, 0.8)
        else if probControl ==#20 17#20 then discrete(0.15, 0.85)
        else if probControl ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95)
    else
        if probAll ==#20 0#20 then discrete(1.0, 0.0)
        else if probAll ==#20 1#20 then discrete(0.95, 0.05)
        else if probAll ==#20 2#20 then discrete(0.9, 0.1)
        else if probAll ==#20 3#20 then discrete(0.85, 0.15)
        else if probAll ==#20 4#20 then discrete(0.8, 0.2)
        else if probAll ==#20 5#20 then discrete(0.75, 0.25)
        else if probAll ==#20 6#20 then discrete(0.7, 0.3)
        else if probAll ==#20 7#20 then discrete(0.65, 0.35)
        else if probAll ==#20 8#20 then discrete(0.6, 0.4)
        else if probAll ==#20 9#20 then discrete(0.55, 0.45)
        else if probAll ==#20 10#20 then discrete(0.5, 0.5)
        else if probAll ==#20 11#20 then discrete(0.45, 0.55)
        else if probAll ==#20 12#20 then discrete(0.4, 0.6)
        else if probAll ==#20 13#20 then discrete(0.35, 0.65)
        else if probAll ==#20 14#20 then discrete(0.3, 0.7)
        else if probAll ==#20 15#20 then discrete(0.25, 0.75)
        else if probAll ==#20 16#20 then discrete(0.2, 0.8)
        else if probAll ==#20 17#20 then discrete(0.15, 0.85)
        else if probAll ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95) in

let treatedGroup7 =
    if isEffective ==#2 1#2 then
        if probTreated ==#20 0#20 then discrete(1.0, 0.0)
        else if probTreated ==#20 1#20 then discrete(0.95, 0.05)
        else if probTreated ==#20 2#20 then discrete(0.9, 0.1)
        else if probTreated ==#20 3#20 then discrete(0.85, 0.15)
        else if probTreated ==#20 4#20 then discrete(0.8, 0.2)
        else if probTreated ==#20 5#20 then discrete(0.75, 0.25)
        else if probTreated ==#20 6#20 then discrete(0.7, 0.3)
        else if probTreated ==#20 7#20 then discrete(0.65, 0.35)
        else if probTreated ==#20 8#20 then discrete(0.6, 0.4)
        else if probTreated ==#20 9#20 then discrete(0.55, 0.45)
        else if probTreated ==#20 10#20 then discrete(0.5, 0.5)
        else if probTreated ==#20 11#20 then discrete(0.45, 0.55)
        else if probTreated ==#20 12#20 then discrete(0.4, 0.6)
        else if probTreated ==#20 13#20 then discrete(0.35, 0.65)
        else if probTreated ==#20 14#20 then discrete(0.3, 0.7)
        else if probTreated ==#20 15#20 then discrete(0.25, 0.75)
```

```
        else if probTreated ==#20 16#20 then discrete(0.2, 0.8)
        else if probTreated ==#20 17#20 then discrete(0.15, 0.85)
        else if probTreated ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95)
    else
        if probAll ==#20 0#20 then discrete(1.0, 0.0)
        else if probAll ==#20 1#20 then discrete(0.95, 0.05)
        else if probAll ==#20 2#20 then discrete(0.9, 0.1)
        else if probAll ==#20 3#20 then discrete(0.85, 0.15)
        else if probAll ==#20 4#20 then discrete(0.8, 0.2)
        else if probAll ==#20 5#20 then discrete(0.75, 0.25)
        else if probAll ==#20 6#20 then discrete(0.7, 0.3)
        else if probAll ==#20 7#20 then discrete(0.65, 0.35)
        else if probAll ==#20 8#20 then discrete(0.6, 0.4)
        else if probAll ==#20 9#20 then discrete(0.55, 0.45)
        else if probAll ==#20 10#20 then discrete(0.5, 0.5)
        else if probAll ==#20 11#20 then discrete(0.45, 0.55)
        else if probAll ==#20 12#20 then discrete(0.4, 0.6)
        else if probAll ==#20 13#20 then discrete(0.35, 0.65)
        else if probAll ==#20 14#20 then discrete(0.3, 0.7)
        else if probAll ==#20 15#20 then discrete(0.25, 0.75)
        else if probAll ==#20 16#20 then discrete(0.2, 0.8)
        else if probAll ==#20 17#20 then discrete(0.15, 0.85)
        else if probAll ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95) in

let controlGroup8 =
    if isEffective ==#2 1#2 then
        if probControl ==#20 0#20 then discrete(1.0, 0.0)
        else if probControl ==#20 1#20 then discrete(0.95, 0.05)
        else if probControl ==#20 2#20 then discrete(0.9, 0.1)
        else if probControl ==#20 3#20 then discrete(0.85, 0.15)
        else if probControl ==#20 4#20 then discrete(0.8, 0.2)
        else if probControl ==#20 5#20 then discrete(0.75, 0.25)
        else if probControl ==#20 6#20 then discrete(0.7, 0.3)
        else if probControl ==#20 7#20 then discrete(0.65, 0.35)
        else if probControl ==#20 8#20 then discrete(0.6, 0.4)
        else if probControl ==#20 9#20 then discrete(0.55, 0.45)
        else if probControl ==#20 10#20 then discrete(0.5, 0.5)
        else if probControl ==#20 11#20 then discrete(0.45, 0.55)
        else if probControl ==#20 12#20 then discrete(0.4, 0.6)
        else if probControl ==#20 13#20 then discrete(0.35, 0.65)
        else if probControl ==#20 14#20 then discrete(0.3, 0.7)
        else if probControl ==#20 15#20 then discrete(0.25, 0.75)
        else if probControl ==#20 16#20 then discrete(0.2, 0.8)
        else if probControl ==#20 17#20 then discrete(0.15, 0.85)
        else if probControl ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95)
    else
        if probAll ==#20 0#20 then discrete(1.0, 0.0)
        else if probAll ==#20 1#20 then discrete(0.95, 0.05)
        else if probAll ==#20 2#20 then discrete(0.9, 0.1)
        else if probAll ==#20 3#20 then discrete(0.85, 0.15)
        else if probAll ==#20 4#20 then discrete(0.8, 0.2)
        else if probAll ==#20 5#20 then discrete(0.75, 0.25)
        else if probAll ==#20 6#20 then discrete(0.7, 0.3)
        else if probAll ==#20 7#20 then discrete(0.65, 0.35)
        else if probAll ==#20 8#20 then discrete(0.6, 0.4)
        else if probAll ==#20 9#20 then discrete(0.55, 0.45)
        else if probAll ==#20 10#20 then discrete(0.5, 0.5)
        else if probAll ==#20 11#20 then discrete(0.45, 0.55)
        else if probAll ==#20 12#20 then discrete(0.4, 0.6)
        else if probAll ==#20 13#20 then discrete(0.35, 0.65)
        else if probAll ==#20 14#20 then discrete(0.3, 0.7)
        else if probAll ==#20 15#20 then discrete(0.25, 0.75)
        else if probAll ==#20 16#20 then discrete(0.2, 0.8)
        else if probAll ==#20 17#20 then discrete(0.15, 0.85)
        else if probAll ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95) in

let treatedGroup8 =
```

```
    if isEffective ==#2 1#2 then
        if probTreated ==#20 0#20 then discrete(1.0, 0.0)
        else if probTreated ==#20 1#20 then discrete(0.95, 0.05)
        else if probTreated ==#20 2#20 then discrete(0.9, 0.1)
        else if probTreated ==#20 3#20 then discrete(0.85, 0.15)
        else if probTreated ==#20 4#20 then discrete(0.8, 0.2)
        else if probTreated ==#20 5#20 then discrete(0.75, 0.25)
        else if probTreated ==#20 6#20 then discrete(0.7, 0.3)
        else if probTreated ==#20 7#20 then discrete(0.65, 0.35)
        else if probTreated ==#20 8#20 then discrete(0.6, 0.4)
        else if probTreated ==#20 9#20 then discrete(0.55, 0.45)
        else if probTreated ==#20 10#20 then discrete(0.5, 0.5)
        else if probTreated ==#20 11#20 then discrete(0.45, 0.55)
        else if probTreated ==#20 12#20 then discrete(0.4, 0.6)
        else if probTreated ==#20 13#20 then discrete(0.35, 0.65)
        else if probTreated ==#20 14#20 then discrete(0.3, 0.7)
        else if probTreated ==#20 15#20 then discrete(0.25, 0.75)
        else if probTreated ==#20 16#20 then discrete(0.2, 0.8)
        else if probTreated ==#20 17#20 then discrete(0.15, 0.85)
        else if probTreated ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95)
    else
        if probAll ==#20 0#20 then discrete(1.0, 0.0)
        else if probAll ==#20 1#20 then discrete(0.95, 0.05)
        else if probAll ==#20 2#20 then discrete(0.9, 0.1)
        else if probAll ==#20 3#20 then discrete(0.85, 0.15)
        else if probAll ==#20 4#20 then discrete(0.8, 0.2)
        else if probAll ==#20 5#20 then discrete(0.75, 0.25)
        else if probAll ==#20 6#20 then discrete(0.7, 0.3)
        else if probAll ==#20 7#20 then discrete(0.65, 0.35)
        else if probAll ==#20 8#20 then discrete(0.6, 0.4)
        else if probAll ==#20 9#20 then discrete(0.55, 0.45)
        else if probAll ==#20 10#20 then discrete(0.5, 0.5)
        else if probAll ==#20 11#20 then discrete(0.45, 0.55)
        else if probAll ==#20 12#20 then discrete(0.4, 0.6)
        else if probAll ==#20 13#20 then discrete(0.35, 0.65)
        else if probAll ==#20 14#20 then discrete(0.3, 0.7)
        else if probAll ==#20 15#20 then discrete(0.25, 0.75)
        else if probAll ==#20 16#20 then discrete(0.2, 0.8)
        else if probAll ==#20 17#20 then discrete(0.15, 0.85)
        else if probAll ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95) in

let controlGroup9 =
    if isEffective ==#2 1#2 then
        if probControl ==#20 0#20 then discrete(1.0, 0.0)
        else if probControl ==#20 1#20 then discrete(0.95, 0.05)
        else if probControl ==#20 2#20 then discrete(0.9, 0.1)
        else if probControl ==#20 3#20 then discrete(0.85, 0.15)
        else if probControl ==#20 4#20 then discrete(0.8, 0.2)
        else if probControl ==#20 5#20 then discrete(0.75, 0.25)
        else if probControl ==#20 6#20 then discrete(0.7, 0.3)
        else if probControl ==#20 7#20 then discrete(0.65, 0.35)
        else if probControl ==#20 8#20 then discrete(0.6, 0.4)
        else if probControl ==#20 9#20 then discrete(0.55, 0.45)
        else if probControl ==#20 10#20 then discrete(0.5, 0.5)
        else if probControl ==#20 11#20 then discrete(0.45, 0.55)
        else if probControl ==#20 12#20 then discrete(0.4, 0.6)
        else if probControl ==#20 13#20 then discrete(0.35, 0.65)
        else if probControl ==#20 14#20 then discrete(0.3, 0.7)
        else if probControl ==#20 15#20 then discrete(0.25, 0.75)
        else if probControl ==#20 16#20 then discrete(0.2, 0.8)
        else if probControl ==#20 17#20 then discrete(0.15, 0.85)
        else if probControl ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95)
    else
        if probAll ==#20 0#20 then discrete(1.0, 0.0)
        else if probAll ==#20 1#20 then discrete(0.95, 0.05)
        else if probAll ==#20 2#20 then discrete(0.9, 0.1)
        else if probAll ==#20 3#20 then discrete(0.85, 0.15)
        else if probAll ==#20 4#20 then discrete(0.8, 0.2)
```

```
        else if probAll ==#20 5#20 then discrete(0.75, 0.25)
        else if probAll ==#20 6#20 then discrete(0.7, 0.3)
        else if probAll ==#20 7#20 then discrete(0.65, 0.35)
        else if probAll ==#20 8#20 then discrete(0.6, 0.4)
        else if probAll ==#20 9#20 then discrete(0.55, 0.45)
        else if probAll ==#20 10#20 then discrete(0.5, 0.5)
        else if probAll ==#20 11#20 then discrete(0.45, 0.55)
        else if probAll ==#20 12#20 then discrete(0.4, 0.6)
        else if probAll ==#20 13#20 then discrete(0.35, 0.65)
        else if probAll ==#20 14#20 then discrete(0.3, 0.7)
        else if probAll ==#20 15#20 then discrete(0.25, 0.75)
        else if probAll ==#20 16#20 then discrete(0.2, 0.8)
        else if probAll ==#20 17#20 then discrete(0.15, 0.85)
        else if probAll ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95) in

let treatedGroup9 =
    if isEffective ==#2 1#2 then
        if probTreated ==#20 0#20 then discrete(1.0, 0.0)
        else if probTreated ==#20 1#20 then discrete(0.95, 0.05)
        else if probTreated ==#20 2#20 then discrete(0.9, 0.1)
        else if probTreated ==#20 3#20 then discrete(0.85, 0.15)
        else if probTreated ==#20 4#20 then discrete(0.8, 0.2)
        else if probTreated ==#20 5#20 then discrete(0.75, 0.25)
        else if probTreated ==#20 6#20 then discrete(0.7, 0.3)
        else if probTreated ==#20 7#20 then discrete(0.65, 0.35)
        else if probTreated ==#20 8#20 then discrete(0.6, 0.4)
        else if probTreated ==#20 9#20 then discrete(0.55, 0.45)
        else if probTreated ==#20 10#20 then discrete(0.5, 0.5)
        else if probTreated ==#20 11#20 then discrete(0.45, 0.55)
        else if probTreated ==#20 12#20 then discrete(0.4, 0.6)
        else if probTreated ==#20 13#20 then discrete(0.35, 0.65)
        else if probTreated ==#20 14#20 then discrete(0.3, 0.7)
        else if probTreated ==#20 15#20 then discrete(0.25, 0.75)
        else if probTreated ==#20 16#20 then discrete(0.2, 0.8)
        else if probTreated ==#20 17#20 then discrete(0.15, 0.85)
        else if probTreated ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95)
    else
        if probAll ==#20 0#20 then discrete(1.0, 0.0)
        else if probAll ==#20 1#20 then discrete(0.95, 0.05)
        else if probAll ==#20 2#20 then discrete(0.9, 0.1)
        else if probAll ==#20 3#20 then discrete(0.85, 0.15)
        else if probAll ==#20 4#20 then discrete(0.8, 0.2)
        else if probAll ==#20 5#20 then discrete(0.75, 0.25)
        else if probAll ==#20 6#20 then discrete(0.7, 0.3)
        else if probAll ==#20 7#20 then discrete(0.65, 0.35)
        else if probAll ==#20 8#20 then discrete(0.6, 0.4)
        else if probAll ==#20 9#20 then discrete(0.55, 0.45)
        else if probAll ==#20 10#20 then discrete(0.5, 0.5)
        else if probAll ==#20 11#20 then discrete(0.45, 0.55)
        else if probAll ==#20 12#20 then discrete(0.4, 0.6)
        else if probAll ==#20 13#20 then discrete(0.35, 0.65)
        else if probAll ==#20 14#20 then discrete(0.3, 0.7)
        else if probAll ==#20 15#20 then discrete(0.25, 0.75)
        else if probAll ==#20 16#20 then discrete(0.2, 0.8)
        else if probAll ==#20 17#20 then discrete(0.15, 0.85)
        else if probAll ==#20 18#20 then discrete(0.1, 0.9)
        else discrete(0.05, 0.95) in

let _0 = observe(controlGroup0 ==#2 0#2) in
let _1 = observe(treatedGroup0 ==#2 0#2) in
let _2 = observe(controlGroup1 ==#2 0#2) in
let _3 = observe(treatedGroup1 ==#2 1#2) in
let _4 = observe(controlGroup2 ==#2 1#2) in
let _5 = observe(treatedGroup2 ==#2 0#2) in
let _6 = observe(controlGroup3 ==#2 1#2) in
let _7 = observe(treatedGroup3 ==#2 0#2) in
let _8 = observe(controlGroup4 ==#2 1#2) in
let _9 = observe(treatedGroup4 ==#2 1#2) in
let _10 = observe(controlGroup5 ==#2 0#2) in
```

```
let _11 = observe(treatedGroup5 ==#2 1#2) in
let _12 = observe(controlGroup6 ==#2 1#2) in
let _13 = observe(treatedGroup6 ==#2 1#2) in
let _14 = observe(controlGroup7 ==#2 1#2) in
let _15 = observe(treatedGroup7 ==#2 1#2) in
let _16 = observe(controlGroup8 ==#2 0#2) in
let _17 = observe(treatedGroup8 ==#2 1#2) in
let _18 = observe(controlGroup9 ==#2 0#2) in
let _19 = observe(treatedGroup9 ==#2 1#2) in

isEffective ==#2 1#2
```

CoinBias

```
let bias = beta(2, 5) in

let tossResults_0 =
    if bias <= 0.05 then discrete(0.95, 0.05)
    else if bias <= 0.1 then discrete(0.9, 0.1)
    else if bias <= 0.15 then discrete(0.85, 0.15)
    else if bias <= 0.2 then discrete(0.8, 0.2)
    else if bias <= 0.25 then discrete(0.75, 0.25)
    else if bias <= 0.3 then discrete(0.7, 0.3)
    else if bias <= 0.35 then discrete(0.65, 0.35)
    else if bias <= 0.4 then discrete(0.6, 0.4)
    else if bias <= 0.45 then discrete(0.55, 0.45)
    else if bias <= 0.5 then discrete(0.5, 0.5)
    else if bias <= 0.55 then discrete(0.45, 0.55)
    else if bias <= 0.6 then discrete(0.4, 0.6)
    else if bias <= 0.65 then discrete(0.35, 0.65)
    else if bias <= 0.7 then discrete(0.3, 0.7)
    else if bias <= 0.75 then discrete(0.25, 0.75)
    else if bias <= 0.8 then discrete(0.2, 0.8)
    else if bias <= 0.85 then discrete(0.15, 0.85)
    else if bias <= 0.9 then discrete(0.1, 0.9)
    else if bias <= 0.95 then discrete(0.05, 0.95)
    else discrete(0.0, 1.0)
in

let tossResults_1 =
    if bias <= 0.05 then discrete(0.95, 0.05)
    else if bias <= 0.1 then discrete(0.9, 0.1)
    else if bias <= 0.15 then discrete(0.85, 0.15)
    else if bias <= 0.2 then discrete(0.8, 0.2)
    else if bias <= 0.25 then discrete(0.75, 0.25)
    else if bias <= 0.3 then discrete(0.7, 0.3)
    else if bias <= 0.35 then discrete(0.65, 0.35)
    else if bias <= 0.4 then discrete(0.6, 0.4)
    else if bias <= 0.45 then discrete(0.55, 0.45)
    else if bias <= 0.5 then discrete(0.5, 0.5)
    else if bias <= 0.55 then discrete(0.45, 0.55)
    else if bias <= 0.6 then discrete(0.4, 0.6)
    else if bias <= 0.65 then discrete(0.35, 0.65)
    else if bias <= 0.7 then discrete(0.3, 0.7)
    else if bias <= 0.75 then discrete(0.25, 0.75)
    else if bias <= 0.8 then discrete(0.2, 0.8)
    else if bias <= 0.85 then discrete(0.15, 0.85)
    else if bias <= 0.9 then discrete(0.1, 0.9)
    else if bias <= 0.95 then discrete(0.05, 0.95)
    else discrete(0.0, 1.0)
in

let tossResults_2 =
    if bias <= 0.05 then discrete(0.95, 0.05)
    else if bias <= 0.1 then discrete(0.9, 0.1)
    else if bias <= 0.15 then discrete(0.85, 0.15)
    else if bias <= 0.2 then discrete(0.8, 0.2)
    else if bias <= 0.25 then discrete(0.75, 0.25)
    else if bias <= 0.3 then discrete(0.7, 0.3)
    else if bias <= 0.35 then discrete(0.65, 0.35)
```

```
    else if bias <= 0.4 then discrete(0.6, 0.4)
    else if bias <= 0.45 then discrete(0.55, 0.45)
    else if bias <= 0.5 then discrete(0.5, 0.5)
    else if bias <= 0.55 then discrete(0.45, 0.55)
    else if bias <= 0.6 then discrete(0.4, 0.6)
    else if bias <= 0.65 then discrete(0.35, 0.65)
    else if bias <= 0.7 then discrete(0.3, 0.7)
    else if bias <= 0.75 then discrete(0.25, 0.75)
    else if bias <= 0.8 then discrete(0.2, 0.8)
    else if bias <= 0.85 then discrete(0.15, 0.85)
    else if bias <= 0.9 then discrete(0.1, 0.9)
    else if bias <= 0.95 then discrete(0.05, 0.95)
    else discrete(0.0, 1.0)
in

let tossResults_3 =
    if bias <= 0.05 then discrete(0.95, 0.05)
    else if bias <= 0.1 then discrete(0.9, 0.1)
    else if bias <= 0.15 then discrete(0.85, 0.15)
    else if bias <= 0.2 then discrete(0.8, 0.2)
    else if bias <= 0.25 then discrete(0.75, 0.25)
    else if bias <= 0.3 then discrete(0.7, 0.3)
    else if bias <= 0.35 then discrete(0.65, 0.35)
    else if bias <= 0.4 then discrete(0.6, 0.4)
    else if bias <= 0.45 then discrete(0.55, 0.45)
    else if bias <= 0.5 then discrete(0.5, 0.5)
    else if bias <= 0.55 then discrete(0.45, 0.55)
    else if bias <= 0.6 then discrete(0.4, 0.6)
    else if bias <= 0.65 then discrete(0.35, 0.65)
    else if bias <= 0.7 then discrete(0.3, 0.7)
    else if bias <= 0.75 then discrete(0.25, 0.75)
    else if bias <= 0.8 then discrete(0.2, 0.8)
    else if bias <= 0.85 then discrete(0.15, 0.85)
    else if bias <= 0.9 then discrete(0.1, 0.9)
    else if bias <= 0.95 then discrete(0.05, 0.95)
    else discrete(0.0, 1.0)
in

let tossResults_4 =
    if bias <= 0.05 then discrete(0.95, 0.05)
    else if bias <= 0.1 then discrete(0.9, 0.1)
    else if bias <= 0.15 then discrete(0.85, 0.15)
    else if bias <= 0.2 then discrete(0.8, 0.2)
    else if bias <= 0.25 then discrete(0.75, 0.25)
    else if bias <= 0.3 then discrete(0.7, 0.3)
    else if bias <= 0.35 then discrete(0.65, 0.35)
    else if bias <= 0.4 then discrete(0.6, 0.4)
    else if bias <= 0.45 then discrete(0.55, 0.45)
    else if bias <= 0.5 then discrete(0.5, 0.5)
    else if bias <= 0.55 then discrete(0.45, 0.55)
    else if bias <= 0.6 then discrete(0.4, 0.6)
    else if bias <= 0.65 then discrete(0.35, 0.65)
    else if bias <= 0.7 then discrete(0.3, 0.7)
    else if bias <= 0.75 then discrete(0.25, 0.75)
    else if bias <= 0.8 then discrete(0.2, 0.8)
    else if bias <= 0.85 then discrete(0.15, 0.85)
    else if bias <= 0.9 then discrete(0.1, 0.9)
    else if bias <= 0.95 then discrete(0.05, 0.95)
    else discrete(0.0, 1.0)
in

let _0 = observe(tossResults_0 ==#2 1#2) in
let _1 = observe(tossResults_1 ==#2 1#2) in
let _2 = observe(tossResults_2 ==#2 0#2) in
let _3 = observe(tossResults_3 ==#2 1#2) in
let _4 = observe(tossResults_4 ==#2 0#2) in

bias < 0.3
```

Election

```
let n = 4000 in
let param = discrete(
  0.01: 250,
  0.01: 251,
  0.01: 252,
  0.01: 253,
  0.01: 254,
  0.01: 255,
  0.01: 256,
  0.01: 257,
  0.01: 258,
  0.01: 259,
  0.01: 260,
  0.01: 261,
  0.01: 262,
  0.01: 263,
  0.01: 264,
  0.01: 265,
  0.01: 266,
  0.01: 267,
  0.01: 268,
  0.01: 269,
  0.01: 270,
  0.01: 271,
  0.01: 272,
  0.01: 273,
  0.01: 274,
  0.01: 275,
  0.01: 276,
  0.01: 277,
  0.01: 278,
  0.01: 279,
  0.01: 280,
  0.01: 281,
  0.01: 282,
  0.01: 283,
  0.01: 284,
  0.01: 285,
  0.01: 286,
  0.01: 287,
  0.01: 288,
  0.01: 289,
  0.01: 290,
  0.01: 291,
  0.01: 292,
  0.01: 293,
  0.01: 294,
  0.01: 295,
  0.01: 296,
  0.01: 297,
  0.01: 298,
  0.01: 299,
  0.01: 300,
  0.01: 301,
  0.01: 302,
  0.01: 303,
  0.01: 304,
  0.01: 305,
  0.01: 306,
  0.01: 307,
  0.01: 308,
  0.01: 309,
  0.01: 310,
  0.01: 311,
  0.01: 312,
  0.01: 313,
  0.01: 314,
  0.01: 315,
  0.01: 316,
  0.01: 317,
  0.01: 318,
```

```
  0.01: 319,
  0.01: 320,
  0.01: 321,
  0.01: 322,
  0.01: 323,
  0.01: 324,
  0.01: 325,
  0.01: 326,
  0.01: 327,
  0.01: 328,
  0.01: 329,
  0.01: 330,
  0.01: 331,
  0.01: 332,
  0.01: 333,
  0.01: 334,
  0.01: 335,
  0.01: 336,
  0.01: 337,
  0.01: 338,
  0.01: 339,
  0.01: 340,
  0.01: 341,
  0.01: 342,
  0.01: 343,
  0.01: 344,
  0.01: 345,
  0.01: 346,
  0.01: 347,
  0.01: 348,
  0.01: 349
) in

let p =
  if param <= 250 then beta(277, 250)
  else if param <= 251 then beta(277, 251)
  else if param <= 252 then beta(277, 252)
  else if param <= 253 then beta(277, 253)
  else if param <= 254 then beta(277, 254)
  else if param <= 255 then beta(277, 255)
  else if param <= 256 then beta(277, 256)
  else if param <= 257 then beta(277, 257)
  else if param <= 258 then beta(277, 258)
  else if param <= 259 then beta(277, 259)
  else if param <= 260 then beta(277, 260)
  else if param <= 261 then beta(277, 261)
  else if param <= 262 then beta(277, 262)
  else if param <= 263 then beta(277, 263)
  else if param <= 264 then beta(277, 264)
  else if param <= 265 then beta(277, 265)
  else if param <= 266 then beta(277, 266)
  else if param <= 267 then beta(277, 267)
  else if param <= 268 then beta(277, 268)
  else if param <= 269 then beta(277, 269)
  else if param <= 270 then beta(277, 270)
  else if param <= 271 then beta(277, 271)
  else if param <= 272 then beta(277, 272)
  else if param <= 273 then beta(277, 273)
  else if param <= 274 then beta(277, 274)
  else if param <= 275 then beta(277, 275)
  else if param <= 276 then beta(277, 276)
  else if param <= 277 then beta(277, 277)
  else if param <= 278 then beta(277, 278)
  else if param <= 279 then beta(277, 279)
  else if param <= 280 then beta(277, 280)
  else if param <= 281 then beta(277, 281)
  else if param <= 282 then beta(277, 282)
  else if param <= 283 then beta(277, 283)
  else if param <= 284 then beta(277, 284)
  else if param <= 285 then beta(277, 285)
  else if param <= 286 then beta(277, 286)
```

```
  else if param <= 287 then beta(277, 287)
  else if param <= 288 then beta(277, 288)
  else if param <= 289 then beta(277, 289)
  else if param <= 290 then beta(277, 290)
  else if param <= 291 then beta(277, 291)
  else if param <= 292 then beta(277, 292)
  else if param <= 293 then beta(277, 293)
  else if param <= 294 then beta(277, 294)
  else if param <= 295 then beta(277, 295)
  else if param <= 296 then beta(277, 296)
  else if param <= 297 then beta(277, 297)
  else if param <= 298 then beta(277, 298)
  else if param <= 299 then beta(277, 299)
  else if param <= 300 then beta(277, 300)
  else if param <= 301 then beta(277, 301)
  else if param <= 302 then beta(277, 302)
  else if param <= 303 then beta(277, 303)
  else if param <= 304 then beta(277, 304)
  else if param <= 305 then beta(277, 305)
  else if param <= 306 then beta(277, 306)
  else if param <= 307 then beta(277, 307)
  else if param <= 308 then beta(277, 308)
  else if param <= 309 then beta(277, 309)
  else if param <= 310 then beta(277, 310)
  else if param <= 311 then beta(277, 311)
  else if param <= 312 then beta(277, 312)
  else if param <= 313 then beta(277, 313)
  else if param <= 314 then beta(277, 314)
  else if param <= 315 then beta(277, 315)
  else if param <= 316 then beta(277, 316)
  else if param <= 317 then beta(277, 317)
  else if param <= 318 then beta(277, 318)
  else if param <= 319 then beta(277, 319)
  else if param <= 320 then beta(277, 320)
  else if param <= 321 then beta(277, 321)
  else if param <= 322 then beta(277, 322)
  else if param <= 323 then beta(277, 323)
  else if param <= 324 then beta(277, 324)
  else if param <= 325 then beta(277, 325)
  else if param <= 326 then beta(277, 326)
  else if param <= 327 then beta(277, 327)
  else if param <= 328 then beta(277, 328)
  else if param <= 329 then beta(277, 329)
  else if param <= 330 then beta(277, 330)
  else if param <= 331 then beta(277, 331)
  else if param <= 332 then beta(277, 332)
  else if param <= 333 then beta(277, 333)
  else if param <= 334 then beta(277, 334)
  else if param <= 335 then beta(277, 335)
  else if param <= 336 then beta(277, 336)
  else if param <= 337 then beta(277, 337)
  else if param <= 338 then beta(277, 338)
  else if param <= 339 then beta(277, 339)
  else if param <= 340 then beta(277, 340)
  else if param <= 341 then beta(277, 341)
  else if param <= 342 then beta(277, 342)
  else if param <= 343 then beta(277, 343)
  else if param <= 344 then beta(277, 344)
  else if param <= 345 then beta(277, 345)
  else if param <= 346 then beta(277, 346)
  else if param <= 347 then beta(277, 347)
  else if param <= 348 then beta(277, 348)
  else beta(277, 349)
in

let votes =
  if p <= 0.05 then binomial(0.025, n)
  else if p <= 0.1 then binomial(0.075, n)
  else if p <= 0.15 then binomial(0.125, n)
  else if p <= 0.2 then binomial(0.175, n)
  else if p <= 0.25 then binomial(0.225, n)
```

```
  else if p <= 0.3 then binomial(0.275, n)
  else if p <= 0.35 then binomial(0.325, n)
  else if p <= 0.4 then binomial(0.375, n)
  else if p <= 0.45 then binomial(0.425, n)
  else if p <= 0.5 then binomial(0.475, n)
  else if p <= 0.55 then binomial(0.525, n)
  else if p <= 0.6 then binomial(0.575, n)
  else if p <= 0.65 then binomial(0.625, n)
  else if p <= 0.7 then binomial(0.675, n)
  else if p <= 0.75 then binomial(0.725, n)
  else if p <= 0.8 then binomial(0.775, n)
  else if p <= 0.85 then binomial(0.825, n)
  else if p <= 0.9 then binomial(0.875, n)
  else if p <= 0.95 then binomial(0.925, n)
  else binomial(0.975, n)
in

let win = votes > 2000 in
let _win = observe(win) in
param <= 250
```

MarkovSwitching (HMM)

```
let seperated = discrete(0.6, 0.4) in

let Z0 =
  if seperated ==#2 0#2 then
    discrete(0.5, 0.5)
  else
    discrete(0.5, 0.5)
in

let X0 =
  if seperated ==#2 0#2 then
    (if Z0 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z0 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y0 =
  if seperated ==#2 0#2 then
    (if Z0 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z0 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z1 =
  if Z0 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X1 =
  if seperated ==#2 0#2 then
    (if Z1 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z1 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y1 =
  if seperated ==#2 0#2 then
    (if Z1 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z1 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z2 =
  if Z1 ==#2 0#2 then
    discrete(0.8, 0.2)
```

```
  else
    discrete(0.2, 0.8)
in

let X2 =
  if seperated ==#2 0#2 then
    (if Z2 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z2 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y2 =
  if seperated ==#2 0#2 then
    (if Z2 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z2 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z3 =
  if Z2 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X3 =
  if seperated ==#2 0#2 then
    (if Z3 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z3 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y3 =
  if seperated ==#2 0#2 then
    (if Z3 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z3 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z4 =
  if Z3 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X4 =
  if seperated ==#2 0#2 then
    (if Z4 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z4 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y4 =
  if seperated ==#2 0#2 then
    (if Z4 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z4 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z5 =
  if Z4 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X5 =
  if seperated ==#2 0#2 then
    (if Z5 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
```

```
    (if Z5 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y5 =
  if seperated ==#2 0#2 then
    (if Z5 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z5 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z6 =
  if Z5 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X6 =
  if seperated ==#2 0#2 then
    (if Z6 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z6 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y6 =
  if seperated ==#2 0#2 then
    (if Z6 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z6 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z7 =
  if Z6 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X7 =
  if seperated ==#2 0#2 then
    (if Z7 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z7 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y7 =
  if seperated ==#2 0#2 then
    (if Z7 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z7 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z8 =
  if Z7 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X8 =
  if seperated ==#2 0#2 then
    (if Z8 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z8 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y8 =
  if seperated ==#2 0#2 then
    (if Z8 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z8 ==#2 0#2 then poisson(3) else poisson(8))
```

```
in

let Z9 =
  if Z8 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X9 =
  if seperated ==#2 0#2 then
    (if Z9 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z9 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y9 =
  if seperated ==#2 0#2 then
    (if Z9 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z9 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z10 =
  if Z9 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X10 =
  if seperated ==#2 0#2 then
    (if Z10 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z10 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y10 =
  if seperated ==#2 0#2 then
    (if Z10 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z10 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z11 =
  if Z10 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X11 =
  if seperated ==#2 0#2 then
    (if Z11 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z11 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y11 =
  if seperated ==#2 0#2 then
    (if Z11 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z11 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z12 =
  if Z11 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in
```

```
let X12 =
  if seperated ==#2 0#2 then
    (if Z12 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z12 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y12 =
  if seperated ==#2 0#2 then
    (if Z12 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z12 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z13 =
  if Z12 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X13 =
  if seperated ==#2 0#2 then
    (if Z13 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z13 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y13 =
  if seperated ==#2 0#2 then
    (if Z13 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z13 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z14 =
  if Z13 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X14 =
  if seperated ==#2 0#2 then
    (if Z14 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z14 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y14 =
  if seperated ==#2 0#2 then
    (if Z14 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z14 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z15 =
  if Z14 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X15 =
  if seperated ==#2 0#2 then
    (if Z15 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z15 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in
```

```
let Y15 =
  if seperated ==#2 0#2 then
    (if Z15 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z15 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z16 =
  if Z15 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X16 =
  if seperated ==#2 0#2 then
    (if Z16 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z16 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y16 =
  if seperated ==#2 0#2 then
    (if Z16 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z16 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z17 =
  if Z16 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X17 =
  if seperated ==#2 0#2 then
    (if Z17 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z17 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y17 =
  if seperated ==#2 0#2 then
    (if Z17 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z17 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z18 =
  if Z17 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X18 =
  if seperated ==#2 0#2 then
    (if Z18 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z18 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y18 =
  if seperated ==#2 0#2 then
    (if Z18 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z18 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z19 =
```

```
  if Z18 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X19 =
  if seperated ==#2 0#2 then
    (if Z19 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z19 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y19 =
  if seperated ==#2 0#2 then
    (if Z19 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z19 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z20 =
  if Z19 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X20 =
  if seperated ==#2 0#2 then
    (if Z20 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z20 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y20 =
  if seperated ==#2 0#2 then
    (if Z20 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z20 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z21 =
  if Z20 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X21 =
  if seperated ==#2 0#2 then
    (if Z21 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z21 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y21 =
  if seperated ==#2 0#2 then
    (if Z21 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z21 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z22 =
  if Z21 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X22 =
  if seperated ==#2 0#2 then
```

```
    (if Z22 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z22 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y22 =
  if seperated ==#2 0#2 then
    (if Z22 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z22 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z23 =
  if Z22 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X23 =
  if seperated ==#2 0#2 then
    (if Z23 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z23 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y23 =
  if seperated ==#2 0#2 then
    (if Z23 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z23 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z24 =
  if Z23 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X24 =
  if seperated ==#2 0#2 then
    (if Z24 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z24 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y24 =
  if seperated ==#2 0#2 then
    (if Z24 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z24 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z25 =
  if Z24 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X25 =
  if seperated ==#2 0#2 then
    (if Z25 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z25 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y25 =
  if seperated ==#2 0#2 then
    (if Z25 ==#2 0#2 then poisson(5) else poisson(8))
```

```
  else
    (if Z25 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z26 =
  if Z25 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X26 =
  if seperated ==#2 0#2 then
    (if Z26 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z26 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y26 =
  if seperated ==#2 0#2 then
    (if Z26 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z26 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z27 =
  if Z26 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X27 =
  if seperated ==#2 0#2 then
    (if Z27 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z27 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y27 =
  if seperated ==#2 0#2 then
    (if Z27 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z27 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z28 =
  if Z27 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X28 =
  if seperated ==#2 0#2 then
    (if Z28 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z28 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y28 =
  if seperated ==#2 0#2 then
    (if Z28 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z28 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z29 =
  if Z28 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
```

```
    discrete(0.2, 0.8)
in

let X29 =
  if seperated ==#2 0#2 then
    (if Z29 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z29 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y29 =
  if seperated ==#2 0#2 then
    (if Z29 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z29 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z30 =
  if Z29 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X30 =
  if seperated ==#2 0#2 then
    (if Z30 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z30 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y30 =
  if seperated ==#2 0#2 then
    (if Z30 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z30 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z31 =
  if Z30 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X31 =
  if seperated ==#2 0#2 then
    (if Z31 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z31 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y31 =
  if seperated ==#2 0#2 then
    (if Z31 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z31 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z32 =
  if Z31 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X32 =
  if seperated ==#2 0#2 then
    (if Z32 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z32 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
```

```
in

let Y32 =
  if seperated ==#2 0#2 then
    (if Z32 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z32 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z33 =
  if Z32 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X33 =
  if seperated ==#2 0#2 then
    (if Z33 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z33 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y33 =
  if seperated ==#2 0#2 then
    (if Z33 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z33 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z34 =
  if Z33 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X34 =
  if seperated ==#2 0#2 then
    (if Z34 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z34 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y34 =
  if seperated ==#2 0#2 then
    (if Z34 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z34 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z35 =
  if Z34 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X35 =
  if seperated ==#2 0#2 then
    (if Z35 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z35 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y35 =
  if seperated ==#2 0#2 then
    (if Z35 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z35 ==#2 0#2 then poisson(3) else poisson(8))
in
```

```
let Z36 =
  if Z35 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X36 =
  if seperated ==#2 0#2 then
    (if Z36 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z36 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y36 =
  if seperated ==#2 0#2 then
    (if Z36 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z36 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z37 =
  if Z36 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X37 =
  if seperated ==#2 0#2 then
    (if Z37 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z37 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y37 =
  if seperated ==#2 0#2 then
    (if Z37 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z37 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z38 =
  if Z37 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X38 =
  if seperated ==#2 0#2 then
    (if Z38 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z38 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y38 =
  if seperated ==#2 0#2 then
    (if Z38 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z38 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z39 =
  if Z38 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in
```

```
let X39 =
  if seperated ==#2 0#2 then
    (if Z39 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z39 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y39 =
  if seperated ==#2 0#2 then
    (if Z39 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z39 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z40 =
  if Z39 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X40 =
  if seperated ==#2 0#2 then
    (if Z40 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z40 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y40 =
  if seperated ==#2 0#2 then
    (if Z40 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z40 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z41 =
  if Z40 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X41 =
  if seperated ==#2 0#2 then
    (if Z41 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z41 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y41 =
  if seperated ==#2 0#2 then
    (if Z41 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z41 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z42 =
  if Z41 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X42 =
  if seperated ==#2 0#2 then
    (if Z42 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z42 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y42 =
```

```
  if seperated ==#2 0#2 then
    (if Z42 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z42 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z43 =
  if Z42 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X43 =
  if seperated ==#2 0#2 then
    (if Z43 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z43 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y43 =
  if seperated ==#2 0#2 then
    (if Z43 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z43 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z44 =
  if Z43 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X44 =
  if seperated ==#2 0#2 then
    (if Z44 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z44 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y44 =
  if seperated ==#2 0#2 then
    (if Z44 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z44 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z45 =
  if Z44 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X45 =
  if seperated ==#2 0#2 then
    (if Z45 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z45 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y45 =
  if seperated ==#2 0#2 then
    (if Z45 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z45 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z46 =
  if Z45 ==#2 0#2 then
```

```
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X46 =
  if seperated ==#2 0#2 then
    (if Z46 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z46 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y46 =
  if seperated ==#2 0#2 then
    (if Z46 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z46 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z47 =
  if Z46 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X47 =
  if seperated ==#2 0#2 then
    (if Z47 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z47 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y47 =
  if seperated ==#2 0#2 then
    (if Z47 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z47 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z48 =
  if Z47 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X48 =
  if seperated ==#2 0#2 then
    (if Z48 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z48 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y48 =
  if seperated ==#2 0#2 then
    (if Z48 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z48 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z49 =
  if Z48 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X49 =
  if seperated ==#2 0#2 then
    (if Z49 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
```

```
  else
    (if Z49 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y49 =
  if seperated ==#2 0#2 then
    (if Z49 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z49 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z50 =
  if Z49 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X50 =
  if seperated ==#2 0#2 then
    (if Z50 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z50 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y50 =
  if seperated ==#2 0#2 then
    (if Z50 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z50 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z51 =
  if Z50 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X51 =
  if seperated ==#2 0#2 then
    (if Z51 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z51 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y51 =
  if seperated ==#2 0#2 then
    (if Z51 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z51 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z52 =
  if Z51 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X52 =
  if seperated ==#2 0#2 then
    (if Z52 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z52 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y52 =
  if seperated ==#2 0#2 then
    (if Z52 ==#2 0#2 then poisson(5) else poisson(8))
  else
```

```
    (if Z52 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z53 =
  if Z52 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X53 =
  if seperated ==#2 0#2 then
    (if Z53 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z53 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y53 =
  if seperated ==#2 0#2 then
    (if Z53 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z53 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z54 =
  if Z53 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X54 =
  if seperated ==#2 0#2 then
    (if Z54 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z54 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y54 =
  if seperated ==#2 0#2 then
    (if Z54 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z54 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z55 =
  if Z54 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X55 =
  if seperated ==#2 0#2 then
    (if Z55 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z55 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y55 =
  if seperated ==#2 0#2 then
    (if Z55 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z55 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z56 =
  if Z55 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
```

```
in

let X56 =
  if seperated ==#2 0#2 then
    (if Z56 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z56 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y56 =
  if seperated ==#2 0#2 then
    (if Z56 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z56 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z57 =
  if Z56 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X57 =
  if seperated ==#2 0#2 then
    (if Z57 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z57 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y57 =
  if seperated ==#2 0#2 then
    (if Z57 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z57 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z58 =
  if Z57 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X58 =
  if seperated ==#2 0#2 then
    (if Z58 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z58 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y58 =
  if seperated ==#2 0#2 then
    (if Z58 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z58 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z59 =
  if Z58 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X59 =
  if seperated ==#2 0#2 then
    (if Z59 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z59 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in
```

```
let Y59 =
  if seperated ==#2 0#2 then
    (if Z59 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z59 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z60 =
  if Z59 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X60 =
  if seperated ==#2 0#2 then
    (if Z60 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z60 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y60 =
  if seperated ==#2 0#2 then
    (if Z60 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z60 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z61 =
  if Z60 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X61 =
  if seperated ==#2 0#2 then
    (if Z61 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z61 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y61 =
  if seperated ==#2 0#2 then
    (if Z61 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z61 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z62 =
  if Z61 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X62 =
  if seperated ==#2 0#2 then
    (if Z62 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z62 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y62 =
  if seperated ==#2 0#2 then
    (if Z62 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z62 ==#2 0#2 then poisson(3) else poisson(8))
in
```

```
let Z63 =
  if Z62 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X63 =
  if seperated ==#2 0#2 then
    (if Z63 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z63 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y63 =
  if seperated ==#2 0#2 then
    (if Z63 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z63 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z64 =
  if Z63 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X64 =
  if seperated ==#2 0#2 then
    (if Z64 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z64 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y64 =
  if seperated ==#2 0#2 then
    (if Z64 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z64 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z65 =
  if Z64 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X65 =
  if seperated ==#2 0#2 then
    (if Z65 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z65 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y65 =
  if seperated ==#2 0#2 then
    (if Z65 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z65 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z66 =
  if Z65 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X66 =
```

```
  if seperated ==#2 0#2 then
    (if Z66 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z66 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y66 =
  if seperated ==#2 0#2 then
    (if Z66 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z66 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z67 =
  if Z66 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X67 =
  if seperated ==#2 0#2 then
    (if Z67 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z67 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y67 =
  if seperated ==#2 0#2 then
    (if Z67 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z67 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z68 =
  if Z67 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X68 =
  if seperated ==#2 0#2 then
    (if Z68 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z68 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y68 =
  if seperated ==#2 0#2 then
    (if Z68 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z68 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z69 =
  if Z68 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X69 =
  if seperated ==#2 0#2 then
    (if Z69 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z69 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y69 =
  if seperated ==#2 0#2 then
```

```
    (if Z69 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z69 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z70 =
  if Z69 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X70 =
  if seperated ==#2 0#2 then
    (if Z70 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z70 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y70 =
  if seperated ==#2 0#2 then
    (if Z70 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z70 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z71 =
  if Z70 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X71 =
  if seperated ==#2 0#2 then
    (if Z71 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z71 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y71 =
  if seperated ==#2 0#2 then
    (if Z71 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z71 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z72 =
  if Z71 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X72 =
  if seperated ==#2 0#2 then
    (if Z72 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z72 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y72 =
  if seperated ==#2 0#2 then
    (if Z72 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z72 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z73 =
  if Z72 ==#2 0#2 then
    discrete(0.8, 0.2)
```

```
  else
    discrete(0.2, 0.8)
in

let X73 =
  if seperated ==#2 0#2 then
    (if Z73 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z73 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y73 =
  if seperated ==#2 0#2 then
    (if Z73 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z73 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z74 =
  if Z73 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X74 =
  if seperated ==#2 0#2 then
    (if Z74 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z74 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y74 =
  if seperated ==#2 0#2 then
    (if Z74 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z74 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z75 =
  if Z74 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X75 =
  if seperated ==#2 0#2 then
    (if Z75 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z75 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y75 =
  if seperated ==#2 0#2 then
    (if Z75 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z75 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z76 =
  if Z75 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X76 =
  if seperated ==#2 0#2 then
    (if Z76 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
```

```
    (if Z76 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y76 =
  if seperated ==#2 0#2 then
    (if Z76 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z76 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z77 =
  if Z76 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X77 =
  if seperated ==#2 0#2 then
    (if Z77 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z77 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y77 =
  if seperated ==#2 0#2 then
    (if Z77 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z77 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z78 =
  if Z77 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X78 =
  if seperated ==#2 0#2 then
    (if Z78 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z78 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y78 =
  if seperated ==#2 0#2 then
    (if Z78 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z78 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z79 =
  if Z78 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X79 =
  if seperated ==#2 0#2 then
    (if Z79 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z79 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y79 =
  if seperated ==#2 0#2 then
    (if Z79 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z79 ==#2 0#2 then poisson(3) else poisson(8))
```

```
in

let Z80 =
  if Z79 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X80 =
  if seperated ==#2 0#2 then
    (if Z80 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z80 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y80 =
  if seperated ==#2 0#2 then
    (if Z80 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z80 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z81 =
  if Z80 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X81 =
  if seperated ==#2 0#2 then
    (if Z81 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z81 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y81 =
  if seperated ==#2 0#2 then
    (if Z81 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z81 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z82 =
  if Z81 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X82 =
  if seperated ==#2 0#2 then
    (if Z82 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z82 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y82 =
  if seperated ==#2 0#2 then
    (if Z82 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z82 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z83 =
  if Z82 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in
```

```
let X83 =
  if seperated ==#2 0#2 then
    (if Z83 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z83 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y83 =
  if seperated ==#2 0#2 then
    (if Z83 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z83 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z84 =
  if Z83 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X84 =
  if seperated ==#2 0#2 then
    (if Z84 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z84 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y84 =
  if seperated ==#2 0#2 then
    (if Z84 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z84 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z85 =
  if Z84 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X85 =
  if seperated ==#2 0#2 then
    (if Z85 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z85 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y85 =
  if seperated ==#2 0#2 then
    (if Z85 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z85 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z86 =
  if Z85 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X86 =
  if seperated ==#2 0#2 then
    (if Z86 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z86 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in
```

```
let Y86 =
  if seperated ==#2 0#2 then
    (if Z86 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z86 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z87 =
  if Z86 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X87 =
  if seperated ==#2 0#2 then
    (if Z87 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z87 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y87 =
  if seperated ==#2 0#2 then
    (if Z87 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z87 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z88 =
  if Z87 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X88 =
  if seperated ==#2 0#2 then
    (if Z88 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z88 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y88 =
  if seperated ==#2 0#2 then
    (if Z88 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z88 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z89 =
  if Z88 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X89 =
  if seperated ==#2 0#2 then
    (if Z89 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z89 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y89 =
  if seperated ==#2 0#2 then
    (if Z89 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z89 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z90 =
```

```
  if Z89 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X90 =
  if seperated ==#2 0#2 then
    (if Z90 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z90 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y90 =
  if seperated ==#2 0#2 then
    (if Z90 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z90 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z91 =
  if Z90 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X91 =
  if seperated ==#2 0#2 then
    (if Z91 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z91 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y91 =
  if seperated ==#2 0#2 then
    (if Z91 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z91 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z92 =
  if Z91 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X92 =
  if seperated ==#2 0#2 then
    (if Z92 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z92 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y92 =
  if seperated ==#2 0#2 then
    (if Z92 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z92 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z93 =
  if Z92 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X93 =
  if seperated ==#2 0#2 then
```

```
    (if Z93 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z93 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y93 =
  if seperated ==#2 0#2 then
    (if Z93 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z93 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z94 =
  if Z93 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X94 =
  if seperated ==#2 0#2 then
    (if Z94 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z94 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y94 =
  if seperated ==#2 0#2 then
    (if Z94 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z94 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z95 =
  if Z94 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X95 =
  if seperated ==#2 0#2 then
    (if Z95 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z95 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y95 =
  if seperated ==#2 0#2 then
    (if Z95 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z95 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z96 =
  if Z95 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X96 =
  if seperated ==#2 0#2 then
    (if Z96 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z96 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y96 =
  if seperated ==#2 0#2 then
    (if Z96 ==#2 0#2 then poisson(5) else poisson(8))
```

```
  else
    (if Z96 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z97 =
  if Z96 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X97 =
  if seperated ==#2 0#2 then
    (if Z97 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z97 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y97 =
  if seperated ==#2 0#2 then
    (if Z97 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z97 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z98 =
  if Z97 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X98 =
  if seperated ==#2 0#2 then
    (if Z98 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z98 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y98 =
  if seperated ==#2 0#2 then
    (if Z98 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z98 ==#2 0#2 then poisson(3) else poisson(8))
in

let Z99 =
  if Z98 ==#2 0#2 then
    discrete(0.8, 0.2)
  else
    discrete(0.2, 0.8)
in

let X99 =
  if seperated ==#2 0#2 then
    (if Z99 ==#2 0#2 then gaussian(5, 1) else gaussian(7, 1))
  else
    (if Z99 ==#2 0#2 then gaussian(5, 1) else gaussian(15, 1))
in

let Y99 =
  if seperated ==#2 0#2 then
    (if Z99 ==#2 0#2 then poisson(5) else poisson(8))
  else
    (if Z99 ==#2 0#2 then poisson(3) else poisson(8))
in

Z0 <#2 1#2
```

StudentInterviews

```
let student_perfect_0 = discrete(0.8, 0.2) in
let student_gpa_0 = if student_perfect_0 ==#2 0#2 then beta(7, 3) else 1.0 in

let student_perfect_1 = discrete(0.8, 0.2) in
let student_gpa_1 = if student_perfect_1 ==#2 0#2 then beta(7, 3) else 1.0 in

let num_recruiters = poisson(25) in
let _recruiters_range = observe(10 <= num_recruiters && num_recruiters <= 40) in

let student_interviews_0 =
    if num_recruiters <= 10 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 10)
        else if student_gpa_0 > 0.875 then binomial(0.6, 10)
        else binomial(0.5, 10)
    else if num_recruiters <= 11 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 11)
        else if student_gpa_0 > 0.875 then binomial(0.6, 11)
        else binomial(0.5, 11)
    else if num_recruiters <= 12 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 12)
        else if student_gpa_0 > 0.875 then binomial(0.6, 12)
        else binomial(0.5, 12)
    else if num_recruiters <= 13 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 13)
        else if student_gpa_0 > 0.875 then binomial(0.6, 13)
        else binomial(0.5, 13)
    else if num_recruiters <= 14 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 14)
        else if student_gpa_0 > 0.875 then binomial(0.6, 14)
        else binomial(0.5, 14)
    else if num_recruiters <= 15 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 15)
        else if student_gpa_0 > 0.875 then binomial(0.6, 15)
        else binomial(0.5, 15)
    else if num_recruiters <= 16 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 16)
        else if student_gpa_0 > 0.875 then binomial(0.6, 16)
        else binomial(0.5, 16)
    else if num_recruiters <= 17 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 17)
        else if student_gpa_0 > 0.875 then binomial(0.6, 17)
        else binomial(0.5, 17)
    else if num_recruiters <= 18 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 18)
        else if student_gpa_0 > 0.875 then binomial(0.6, 18)
        else binomial(0.5, 18)
    else if num_recruiters <= 19 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 19)
        else if student_gpa_0 > 0.875 then binomial(0.6, 19)
        else binomial(0.5, 19)
    else if num_recruiters <= 20 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 20)
        else if student_gpa_0 > 0.875 then binomial(0.6, 20)
        else binomial(0.5, 20)
    else if num_recruiters <= 21 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 21)
        else if student_gpa_0 > 0.875 then binomial(0.6, 21)
        else binomial(0.5, 21)
    else if num_recruiters <= 22 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 22)
        else if student_gpa_0 > 0.875 then binomial(0.6, 22)
        else binomial(0.5, 22)
    else if num_recruiters <= 23 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 23)
        else if student_gpa_0 > 0.875 then binomial(0.6, 23)
        else binomial(0.5, 23)
    else if num_recruiters <= 24 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 24)
        else if student_gpa_0 > 0.875 then binomial(0.6, 24)
        else binomial(0.5, 24)
    else if num_recruiters <= 25 then
```

```
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 25)
        else if student_gpa_0 > 0.875 then binomial(0.6, 25)
        else binomial(0.5, 25)
    else if num_recruiters <= 26 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 26)
        else if student_gpa_0 > 0.875 then binomial(0.6, 26)
        else binomial(0.5, 26)
    else if num_recruiters <= 27 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 27)
        else if student_gpa_0 > 0.875 then binomial(0.6, 27)
        else binomial(0.5, 27)
    else if num_recruiters <= 28 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 28)
        else if student_gpa_0 > 0.875 then binomial(0.6, 28)
        else binomial(0.5, 28)
    else if num_recruiters <= 29 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 29)
        else if student_gpa_0 > 0.875 then binomial(0.6, 29)
        else binomial(0.5, 29)
    else if num_recruiters <= 30 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 30)
        else if student_gpa_0 > 0.875 then binomial(0.6, 30)
        else binomial(0.5, 30)
    else if num_recruiters <= 31 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 31)
        else if student_gpa_0 > 0.875 then binomial(0.6, 31)
        else binomial(0.5, 31)
    else if num_recruiters <= 32 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 32)
        else if student_gpa_0 > 0.875 then binomial(0.6, 32)
        else binomial(0.5, 32)
    else if num_recruiters <= 33 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 33)
        else if student_gpa_0 > 0.875 then binomial(0.6, 33)
        else binomial(0.5, 33)
    else if num_recruiters <= 34 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 34)
        else if student_gpa_0 > 0.875 then binomial(0.6, 34)
        else binomial(0.5, 34)
    else if num_recruiters <= 35 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 35)
        else if student_gpa_0 > 0.875 then binomial(0.6, 35)
        else binomial(0.5, 35)
    else if num_recruiters <= 36 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 36)
        else if student_gpa_0 > 0.875 then binomial(0.6, 36)
        else binomial(0.5, 36)
    else if num_recruiters <= 37 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 37)
        else if student_gpa_0 > 0.875 then binomial(0.6, 37)
        else binomial(0.5, 37)
    else if num_recruiters <= 38 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 38)
        else if student_gpa_0 > 0.875 then binomial(0.6, 38)
        else binomial(0.5, 38)
    else if num_recruiters <= 39 then
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 39)
        else if student_gpa_0 > 0.875 then binomial(0.6, 39)
        else binomial(0.5, 39)
    else
        if student_perfect_0 ==#2 1#2 then binomial(0.9, 40)
        else if student_gpa_0 > 0.875 then binomial(0.6, 40)
        else binomial(0.5, 40)
in

let student_interviews_1 =
    if num_recruiters <= 10 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 10)
        else if student_gpa_1 > 0.875 then binomial(0.6, 10)
        else binomial(0.5, 10)
    else if num_recruiters <= 11 then
```

```
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 11)
        else if student_gpa_1 > 0.875 then binomial(0.6, 11)
        else binomial(0.5, 11)
    else if num_recruiters <= 12 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 12)
        else if student_gpa_1 > 0.875 then binomial(0.6, 12)
        else binomial(0.5, 12)
    else if num_recruiters <= 13 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 13)
        else if student_gpa_1 > 0.875 then binomial(0.6, 13)
        else binomial(0.5, 13)
    else if num_recruiters <= 14 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 14)
        else if student_gpa_1 > 0.875 then binomial(0.6, 14)
        else binomial(0.5, 14)
    else if num_recruiters <= 15 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 15)
        else if student_gpa_1 > 0.875 then binomial(0.6, 15)
        else binomial(0.5, 15)
    else if num_recruiters <= 16 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 16)
        else if student_gpa_1 > 0.875 then binomial(0.6, 16)
        else binomial(0.5, 16)
    else if num_recruiters <= 17 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 17)
        else if student_gpa_1 > 0.875 then binomial(0.6, 17)
        else binomial(0.5, 17)
    else if num_recruiters <= 18 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 18)
        else if student_gpa_1 > 0.875 then binomial(0.6, 18)
        else binomial(0.5, 18)
    else if num_recruiters <= 19 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 19)
        else if student_gpa_1 > 0.875 then binomial(0.6, 19)
        else binomial(0.5, 19)
    else if num_recruiters <= 20 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 20)
        else if student_gpa_1 > 0.875 then binomial(0.6, 20)
        else binomial(0.5, 20)
    else if num_recruiters <= 21 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 21)
        else if student_gpa_1 > 0.875 then binomial(0.6, 21)
        else binomial(0.5, 21)
    else if num_recruiters <= 22 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 22)
        else if student_gpa_1 > 0.875 then binomial(0.6, 22)
        else binomial(0.5, 22)
    else if num_recruiters <= 23 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 23)
        else if student_gpa_1 > 0.875 then binomial(0.6, 23)
        else binomial(0.5, 23)
    else if num_recruiters <= 24 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 24)
        else if student_gpa_1 > 0.875 then binomial(0.6, 24)
        else binomial(0.5, 24)
    else if num_recruiters <= 25 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 25)
        else if student_gpa_1 > 0.875 then binomial(0.6, 25)
        else binomial(0.5, 25)
    else if num_recruiters <= 26 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 26)
        else if student_gpa_1 > 0.875 then binomial(0.6, 26)
        else binomial(0.5, 26)
    else if num_recruiters <= 27 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 27)
        else if student_gpa_1 > 0.875 then binomial(0.6, 27)
        else binomial(0.5, 27)
    else if num_recruiters <= 28 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 28)
        else if student_gpa_1 > 0.875 then binomial(0.6, 28)
        else binomial(0.5, 28)
```

```
    else if num_recruiters <= 29 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 29)
        else if student_gpa_1 > 0.875 then binomial(0.6, 29)
        else binomial(0.5, 29)
    else if num_recruiters <= 30 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 30)
        else if student_gpa_1 > 0.875 then binomial(0.6, 30)
        else binomial(0.5, 30)
    else if num_recruiters <= 31 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 31)
        else if student_gpa_1 > 0.875 then binomial(0.6, 31)
        else binomial(0.5, 31)
    else if num_recruiters <= 32 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 32)
        else if student_gpa_1 > 0.875 then binomial(0.6, 32)
        else binomial(0.5, 32)
    else if num_recruiters <= 33 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 33)
        else if student_gpa_1 > 0.875 then binomial(0.6, 33)
        else binomial(0.5, 33)
    else if num_recruiters <= 34 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 34)
        else if student_gpa_1 > 0.875 then binomial(0.6, 34)
        else binomial(0.5, 34)
    else if num_recruiters <= 35 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 35)
        else if student_gpa_1 > 0.875 then binomial(0.6, 35)
        else binomial(0.5, 35)
    else if num_recruiters <= 36 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 36)
        else if student_gpa_1 > 0.875 then binomial(0.6, 36)
        else binomial(0.5, 36)
    else if num_recruiters <= 37 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 37)
        else if student_gpa_1 > 0.875 then binomial(0.6, 37)
        else binomial(0.5, 37)
    else if num_recruiters <= 38 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 38)
        else if student_gpa_1 > 0.875 then binomial(0.6, 38)
        else binomial(0.5, 38)
    else if num_recruiters <= 39 then
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 39)
        else if student_gpa_1 > 0.875 then binomial(0.6, 39)
        else binomial(0.5, 39)
    else
        if student_perfect_1 ==#2 1#2 then binomial(0.9, 40)
        else if student_gpa_1 > 0.875 then binomial(0.6, 40)
        else binomial(0.5, 40)
in

let student_offers_0 =
    if student_interviews_0 <= 0 then binomial(0.4, 0)
    else if student_interviews_0 <= 1 then binomial(0.4, 1)
    else if student_interviews_0 <= 2 then binomial(0.4, 2)
    else if student_interviews_0 <= 3 then binomial(0.4, 3)
    else if student_interviews_0 <= 4 then binomial(0.4, 4)
    else if student_interviews_0 <= 5 then binomial(0.4, 5)
    else if student_interviews_0 <= 6 then binomial(0.4, 6)
    else if student_interviews_0 <= 7 then binomial(0.4, 7)
    else if student_interviews_0 <= 8 then binomial(0.4, 8)
    else if student_interviews_0 <= 9 then binomial(0.4, 9)
    else if student_interviews_0 <= 10 then binomial(0.4, 10)
    else if student_interviews_0 <= 11 then binomial(0.4, 11)
    else if student_interviews_0 <= 12 then binomial(0.4, 12)
    else if student_interviews_0 <= 13 then binomial(0.4, 13)
    else if student_interviews_0 <= 14 then binomial(0.4, 14)
    else if student_interviews_0 <= 15 then binomial(0.4, 15)
    else if student_interviews_0 <= 16 then binomial(0.4, 16)
    else if student_interviews_0 <= 17 then binomial(0.4, 17)
    else if student_interviews_0 <= 18 then binomial(0.4, 18)
    else if student_interviews_0 <= 19 then binomial(0.4, 19)
```

```
    else if student_interviews_0 <= 20 then binomial(0.4, 20)
    else if student_interviews_0 <= 21 then binomial(0.4, 21)
    else if student_interviews_0 <= 22 then binomial(0.4, 22)
    else if student_interviews_0 <= 23 then binomial(0.4, 23)
    else if student_interviews_0 <= 24 then binomial(0.4, 24)
    else if student_interviews_0 <= 25 then binomial(0.4, 25)
    else if student_interviews_0 <= 26 then binomial(0.4, 26)
    else if student_interviews_0 <= 27 then binomial(0.4, 27)
    else if student_interviews_0 <= 28 then binomial(0.4, 28)
    else if student_interviews_0 <= 29 then binomial(0.4, 29)
    else if student_interviews_0 <= 30 then binomial(0.4, 30)
    else if student_interviews_0 <= 31 then binomial(0.4, 31)
    else if student_interviews_0 <= 32 then binomial(0.4, 32)
    else if student_interviews_0 <= 33 then binomial(0.4, 33)
    else if student_interviews_0 <= 34 then binomial(0.4, 34)
    else if student_interviews_0 <= 35 then binomial(0.4, 35)
    else if student_interviews_0 <= 36 then binomial(0.4, 36)
    else if student_interviews_0 <= 37 then binomial(0.4, 37)
    else if student_interviews_0 <= 38 then binomial(0.4, 38)
    else if student_interviews_0 <= 39 then binomial(0.4, 39)
    else binomial(0.4, 40)
in

let student_offers_1 =
    if student_interviews_1 <= 0 then binomial(0.4, 0)
    else if student_interviews_1 <= 1 then binomial(0.4, 1)
    else if student_interviews_1 <= 2 then binomial(0.4, 2)
    else if student_interviews_1 <= 3 then binomial(0.4, 3)
    else if student_interviews_1 <= 4 then binomial(0.4, 4)
    else if student_interviews_1 <= 5 then binomial(0.4, 5)
    else if student_interviews_1 <= 6 then binomial(0.4, 6)
    else if student_interviews_1 <= 7 then binomial(0.4, 7)
    else if student_interviews_1 <= 8 then binomial(0.4, 8)
    else if student_interviews_1 <= 9 then binomial(0.4, 9)
    else if student_interviews_1 <= 10 then binomial(0.4, 10)
    else if student_interviews_1 <= 11 then binomial(0.4, 11)
    else if student_interviews_1 <= 12 then binomial(0.4, 12)
    else if student_interviews_1 <= 13 then binomial(0.4, 13)
    else if student_interviews_1 <= 14 then binomial(0.4, 14)
    else if student_interviews_1 <= 15 then binomial(0.4, 15)
    else if student_interviews_1 <= 16 then binomial(0.4, 16)
    else if student_interviews_1 <= 17 then binomial(0.4, 17)
    else if student_interviews_1 <= 18 then binomial(0.4, 18)
    else if student_interviews_1 <= 19 then binomial(0.4, 19)
    else if student_interviews_1 <= 20 then binomial(0.4, 20)
    else if student_interviews_1 <= 21 then binomial(0.4, 21)
    else if student_interviews_1 <= 22 then binomial(0.4, 22)
    else if student_interviews_1 <= 23 then binomial(0.4, 23)
    else if student_interviews_1 <= 24 then binomial(0.4, 24)
    else if student_interviews_1 <= 25 then binomial(0.4, 25)
    else if student_interviews_1 <= 26 then binomial(0.4, 26)
    else if student_interviews_1 <= 27 then binomial(0.4, 27)
    else if student_interviews_1 <= 28 then binomial(0.4, 28)
    else if student_interviews_1 <= 29 then binomial(0.4, 29)
    else if student_interviews_1 <= 30 then binomial(0.4, 30)
    else if student_interviews_1 <= 31 then binomial(0.4, 31)
    else if student_interviews_1 <= 32 then binomial(0.4, 32)
    else if student_interviews_1 <= 33 then binomial(0.4, 33)
    else if student_interviews_1 <= 34 then binomial(0.4, 34)
    else if student_interviews_1 <= 35 then binomial(0.4, 35)
    else if student_interviews_1 <= 36 then binomial(0.4, 36)
    else if student_interviews_1 <= 37 then binomial(0.4, 37)
    else if student_interviews_1 <= 38 then binomial(0.4, 38)
    else if student_interviews_1 <= 39 then binomial(0.4, 39)
    else binomial(0.4, 40)
in

let _offer = observe(1 <= student_offers_0 && student_offers_0 <= 1) in
let _recruiters_positive = observe(num_recruiters > 10) in

student_gpa_0 < 0.125
```

SurveyUnbias

```
let bias_0 = beta(1, 1) in
let bias_1 = beta(1, 1) in

let votes_0 =
    if bias_0 <= 0.05 then binomial(0.05, 1000)
    else if bias_0 <= 0.1 then binomial(0.1, 1000)
    else if bias_0 <= 0.15 then binomial(0.15, 1000)
    else if bias_0 <= 0.2 then binomial(0.2, 1000)
    else if bias_0 <= 0.25 then binomial(0.25, 1000)
    else if bias_0 <= 0.3 then binomial(0.3, 1000)
    else if bias_0 <= 0.35 then binomial(0.35, 1000)
    else if bias_0 <= 0.4 then binomial(0.4, 1000)
    else if bias_0 <= 0.45 then binomial(0.45, 1000)
    else if bias_0 <= 0.5 then binomial(0.5, 1000)
    else if bias_0 <= 0.55 then binomial(0.55, 1000)
    else if bias_0 <= 0.6 then binomial(0.6, 1000)
    else if bias_0 <= 0.65 then binomial(0.65, 1000)
    else if bias_0 <= 0.7 then binomial(0.7, 1000)
    else if bias_0 <= 0.75 then binomial(0.75, 1000)
    else if bias_0 <= 0.8 then binomial(0.8, 1000)
    else if bias_0 <= 0.85 then binomial(0.85, 1000)
    else if bias_0 <= 0.9 then binomial(0.9, 1000)
    else if bias_0 <= 0.95 then binomial(0.95, 1000)
    else binomial(1.0, 1000)
in

let votes_1 =
    if bias_1 <= 0.05 then binomial(0.05, 2000)
    else if bias_1 <= 0.1 then binomial(0.1, 2000)
    else if bias_1 <= 0.15 then binomial(0.15, 2000)
    else if bias_1 <= 0.2 then binomial(0.2, 2000)
    else if bias_1 <= 0.25 then binomial(0.25, 2000)
    else if bias_1 <= 0.3 then binomial(0.3, 2000)
    else if bias_1 <= 0.35 then binomial(0.35, 2000)
    else if bias_1 <= 0.4 then binomial(0.4, 2000)
    else if bias_1 <= 0.45 then binomial(0.45, 2000)
    else if bias_1 <= 0.5 then binomial(0.5, 2000)
    else if bias_1 <= 0.55 then binomial(0.55, 2000)
    else if bias_1 <= 0.6 then binomial(0.6, 2000)
    else if bias_1 <= 0.65 then binomial(0.65, 2000)
    else if bias_1 <= 0.7 then binomial(0.7, 2000)
    else if bias_1 <= 0.75 then binomial(0.75, 2000)
    else if bias_1 <= 0.8 then binomial(0.8, 2000)
    else if bias_1 <= 0.85 then binomial(0.85, 2000)
    else if bias_1 <= 0.9 then binomial(0.9, 2000)
    else if bias_1 <= 0.95 then binomial(0.95, 2000)
    else binomial(1.0, 2000)
in

let ansBias_0 = bias_0 in
let ansBias_1 = bias_1 in
let ansBias_2 = bias_0 in
let ansBias_3 = bias_1 in
let ansBias_4 = bias_0 in

let answer_0 =
    if ansBias_0 <= 0.05 then discrete(0.95, 0.05)
    else if ansBias_0 <= 0.1 then discrete(0.9, 0.1)
    else if ansBias_0 <= 0.15 then discrete(0.85, 0.15)
    else if ansBias_0 <= 0.2 then discrete(0.8, 0.2)
    else if ansBias_0 <= 0.25 then discrete(0.75, 0.25)
    else if ansBias_0 <= 0.3 then discrete(0.7, 0.3)
    else if ansBias_0 <= 0.35 then discrete(0.65, 0.35)
    else if ansBias_0 <= 0.4 then discrete(0.6, 0.4)
    else if ansBias_0 <= 0.45 then discrete(0.55, 0.45)
    else if ansBias_0 <= 0.5 then discrete(0.5, 0.5)
    else if ansBias_0 <= 0.55 then discrete(0.45, 0.55)
    else if ansBias_0 <= 0.6 then discrete(0.4, 0.6)
    else if ansBias_0 <= 0.65 then discrete(0.35, 0.65)
    else if ansBias_0 <= 0.7 then discrete(0.3, 0.7)
```

```
    else if ansBias_0 <= 0.75 then discrete(0.25, 0.75)
    else if ansBias_0 <= 0.8 then discrete(0.2, 0.8)
    else if ansBias_0 <= 0.85 then discrete(0.15, 0.85)
    else if ansBias_0 <= 0.9 then discrete(0.1, 0.9)
    else if ansBias_0 <= 0.95 then discrete(0.05, 0.95)
    else discrete(0.0, 1.0)
in

let answer_1 =
    if ansBias_1 <= 0.05 then discrete(0.95, 0.05)
    else if ansBias_1 <= 0.1 then discrete(0.9, 0.1)
    else if ansBias_1 <= 0.15 then discrete(0.85, 0.15)
    else if ansBias_1 <= 0.2 then discrete(0.8, 0.2)
    else if ansBias_1 <= 0.25 then discrete(0.75, 0.25)
    else if ansBias_1 <= 0.3 then discrete(0.7, 0.3)
    else if ansBias_1 <= 0.35 then discrete(0.65, 0.35)
    else if ansBias_1 <= 0.4 then discrete(0.6, 0.4)
    else if ansBias_1 <= 0.45 then discrete(0.55, 0.45)
    else if ansBias_1 <= 0.5 then discrete(0.5, 0.5)
    else if ansBias_1 <= 0.55 then discrete(0.45, 0.55)
    else if ansBias_1 <= 0.6 then discrete(0.4, 0.6)
    else if ansBias_1 <= 0.65 then discrete(0.35, 0.65)
    else if ansBias_1 <= 0.7 then discrete(0.3, 0.7)
    else if ansBias_1 <= 0.75 then discrete(0.25, 0.75)
    else if ansBias_1 <= 0.8 then discrete(0.2, 0.8)
    else if ansBias_1 <= 0.85 then discrete(0.15, 0.85)
    else if ansBias_1 <= 0.9 then discrete(0.1, 0.9)
    else if ansBias_1 <= 0.95 then discrete(0.05, 0.95)
    else discrete(0.0, 1.0)
in

let answer_2 =
    if ansBias_2 <= 0.05 then discrete(0.95, 0.05)
    else if ansBias_2 <= 0.1 then discrete(0.9, 0.1)
    else if ansBias_2 <= 0.15 then discrete(0.85, 0.15)
    else if ansBias_2 <= 0.2 then discrete(0.8, 0.2)
    else if ansBias_2 <= 0.25 then discrete(0.75, 0.25)
    else if ansBias_2 <= 0.3 then discrete(0.7, 0.3)
    else if ansBias_2 <= 0.35 then discrete(0.65, 0.35)
    else if ansBias_2 <= 0.4 then discrete(0.6, 0.4)
    else if ansBias_2 <= 0.45 then discrete(0.55, 0.45)
    else if ansBias_2 <= 0.5 then discrete(0.5, 0.5)
    else if ansBias_2 <= 0.55 then discrete(0.45, 0.55)
    else if ansBias_2 <= 0.6 then discrete(0.4, 0.6)
    else if ansBias_2 <= 0.65 then discrete(0.35, 0.65)
    else if ansBias_2 <= 0.7 then discrete(0.3, 0.7)
    else if ansBias_2 <= 0.75 then discrete(0.25, 0.75)
    else if ansBias_2 <= 0.8 then discrete(0.2, 0.8)
    else if ansBias_2 <= 0.85 then discrete(0.15, 0.85)
    else if ansBias_2 <= 0.9 then discrete(0.1, 0.9)
    else if ansBias_2 <= 0.95 then discrete(0.05, 0.95)
    else discrete(0.0, 1.0)
in

let answer_3 =
    if ansBias_3 <= 0.05 then discrete(0.95, 0.05)
    else if ansBias_3 <= 0.1 then discrete(0.9, 0.1)
    else if ansBias_3 <= 0.15 then discrete(0.85, 0.15)
    else if ansBias_3 <= 0.2 then discrete(0.8, 0.2)
    else if ansBias_3 <= 0.25 then discrete(0.75, 0.25)
    else if ansBias_3 <= 0.3 then discrete(0.7, 0.3)
    else if ansBias_3 <= 0.35 then discrete(0.65, 0.35)
    else if ansBias_3 <= 0.4 then discrete(0.6, 0.4)
    else if ansBias_3 <= 0.45 then discrete(0.55, 0.45)
    else if ansBias_3 <= 0.5 then discrete(0.5, 0.5)
    else if ansBias_3 <= 0.55 then discrete(0.45, 0.55)
    else if ansBias_3 <= 0.6 then discrete(0.4, 0.6)
    else if ansBias_3 <= 0.65 then discrete(0.35, 0.65)
    else if ansBias_3 <= 0.7 then discrete(0.3, 0.7)
    else if ansBias_3 <= 0.75 then discrete(0.25, 0.75)
    else if ansBias_3 <= 0.8 then discrete(0.2, 0.8)
```

```
    else if ansBias_3 <= 0.85 then discrete(0.15, 0.85)
    else if ansBias_3 <= 0.9 then discrete(0.1, 0.9)
    else if ansBias_3 <= 0.95 then discrete(0.05, 0.95)
    else discrete(0.0, 1.0)
in

let answer_4 =
    if ansBias_4 <= 0.05 then discrete(0.95, 0.05)
    else if ansBias_4 <= 0.1 then discrete(0.9, 0.1)
    else if ansBias_4 <= 0.15 then discrete(0.85, 0.15)
    else if ansBias_4 <= 0.2 then discrete(0.8, 0.2)
    else if ansBias_4 <= 0.25 then discrete(0.75, 0.25)
    else if ansBias_4 <= 0.3 then discrete(0.7, 0.3)
    else if ansBias_4 <= 0.35 then discrete(0.65, 0.35)
    else if ansBias_4 <= 0.4 then discrete(0.6, 0.4)
    else if ansBias_4 <= 0.45 then discrete(0.55, 0.45)
    else if ansBias_4 <= 0.5 then discrete(0.5, 0.5)
    else if ansBias_4 <= 0.55 then discrete(0.45, 0.55)
    else if ansBias_4 <= 0.6 then discrete(0.4, 0.6)
    else if ansBias_4 <= 0.65 then discrete(0.35, 0.65)
    else if ansBias_4 <= 0.7 then discrete(0.3, 0.7)
    else if ansBias_4 <= 0.75 then discrete(0.25, 0.75)
    else if ansBias_4 <= 0.8 then discrete(0.2, 0.8)
    else if ansBias_4 <= 0.85 then discrete(0.15, 0.85)
    else if ansBias_4 <= 0.9 then discrete(0.1, 0.9)
    else if ansBias_4 <= 0.95 then discrete(0.05, 0.95)
    else discrete(0.0, 1.0)
in

let _0 = observe(answer_0 ==#2 1#2) in
let _1 = observe(answer_1 ==#2 0#2) in
let _2 = observe(answer_2 ==#2 1#2) in
let _3 = observe(answer_3 ==#2 1#2) in
let _4 = observe(answer_4 ==#2 0#2) in

bias_0 < 0.5
```

TrueSkill

```
let perfB1 = binomial(0.9, 100) in
let _perfB1_observed = observe(80 <= perfB1) in

let skillA = poisson(100) in
let _skillA_observed = observe(77 <= skillA && skillA < 125) in

let perfA1 =
  if skillA <= 77 then binomial(0.9, 77)
  else if skillA <= 78 then binomial(0.9, 78)
  else if skillA <= 79 then binomial(0.9, 79)
  else if skillA <= 80 then binomial(0.9, 80)
  else if skillA <= 81 then binomial(0.9, 81)
  else if skillA <= 82 then binomial(0.9, 82)
  else if skillA <= 83 then binomial(0.9, 83)
  else if skillA <= 84 then binomial(0.9, 84)
  else if skillA <= 85 then binomial(0.9, 85)
  else if skillA <= 86 then binomial(0.9, 86)
  else if skillA <= 87 then binomial(0.9, 87)
  else if skillA <= 88 then binomial(0.9, 88)
  else if skillA <= 89 then binomial(0.9, 89)
  else if skillA <= 90 then binomial(0.9, 90)
  else if skillA <= 91 then binomial(0.9, 91)
  else if skillA <= 92 then binomial(0.9, 92)
  else if skillA <= 93 then binomial(0.9, 93)
  else if skillA <= 94 then binomial(0.9, 94)
  else if skillA <= 95 then binomial(0.9, 95)
  else if skillA <= 96 then binomial(0.9, 96)
  else if skillA <= 97 then binomial(0.9, 97)
  else if skillA <= 98 then binomial(0.9, 98)
  else if skillA <= 99 then binomial(0.9, 99)
  else if skillA <= 100 then binomial(0.9, 100)
```

```
  else if skillA <= 101 then binomial(0.9, 101)
  else if skillA <= 102 then binomial(0.9, 102)
  else if skillA <= 103 then binomial(0.9, 103)
  else if skillA <= 104 then binomial(0.9, 104)
  else if skillA <= 105 then binomial(0.9, 105)
  else if skillA <= 106 then binomial(0.9, 106)
  else if skillA <= 107 then binomial(0.9, 107)
  else if skillA <= 108 then binomial(0.9, 108)
  else if skillA <= 109 then binomial(0.9, 109)
  else if skillA <= 110 then binomial(0.9, 110)
  else if skillA <= 111 then binomial(0.9, 111)
  else if skillA <= 112 then binomial(0.9, 112)
  else if skillA <= 113 then binomial(0.9, 113)
  else if skillA <= 114 then binomial(0.9, 114)
  else if skillA <= 115 then binomial(0.9, 115)
  else if skillA <= 116 then binomial(0.9, 116)
  else if skillA <= 117 then binomial(0.9, 117)
  else if skillA <= 118 then binomial(0.9, 118)
  else if skillA <= 119 then binomial(0.9, 119)
  else if skillA <= 120 then binomial(0.9, 120)
  else if skillA <= 121 then binomial(0.9, 121)
  else if skillA <= 122 then binomial(0.9, 122)
  else if skillA <= 123 then binomial(0.9, 123)
  else binomial(0.9, 124)
in

let result =
  if perfB1 <= 80 then
    if perfA1 > 80 then 1.0 else 0.0
  else if perfB1 <= 81 then
    if perfA1 > 81 then 1.0 else 0.0
  else if perfB1 <= 82 then
    if perfA1 > 82 then 1.0 else 0.0
  else if perfB1 <= 83 then
    if perfA1 > 83 then 1.0 else 0.0
  else if perfB1 <= 84 then
    if perfA1 > 84 then 1.0 else 0.0
  else if perfB1 <= 85 then
    if perfA1 > 85 then 1.0 else 0.0
  else if perfB1 <= 86 then
    if perfA1 > 86 then 1.0 else 0.0
  else if perfB1 <= 87 then
    if perfA1 > 87 then 1.0 else 0.0
  else if perfB1 <= 88 then
    if perfA1 > 88 then 1.0 else 0.0
  else if perfB1 <= 89 then
    if perfA1 > 89 then 1.0 else 0.0
  else if perfB1 <= 90 then
    if perfA1 > 90 then 1.0 else 0.0
  else if perfB1 <= 91 then
    if perfA1 > 91 then 1.0 else 0.0
  else if perfB1 <= 92 then
    if perfA1 > 92 then 1.0 else 0.0
  else if perfB1 <= 93 then
    if perfA1 > 93 then 1.0 else 0.0
  else if perfB1 <= 94 then
    if perfA1 > 94 then 1.0 else 0.0
  else if perfB1 <= 95 then
    if perfA1 > 95 then 1.0 else 0.0
  else if perfB1 <= 96 then
    if perfA1 > 96 then 1.0 else 0.0
  else if perfB1 <= 97 then
    if perfA1 > 97 then 1.0 else 0.0
  else if perfB1 <= 98 then
    if perfA1 > 98 then 1.0 else 0.0
  else if perfB1 <= 99 then
    if perfA1 > 99 then 1.0 else 0.0
  else
    if perfA1 > 100 then 1.0 else 0.0
in

```

```
103 let _result_observed = observe(result <= 0 && result >= 0) in
104 50 <= perfA1 && perfA1 <= 50
```

Received 2026-03-17; accepted 2026-06-10