A Syntax of Core Calculus

The metalanguage in this paper follows that of Turbak et al. [63, Appx. A]. For completeness, Lst. 9 shows the syntax of the core calculus whose semantics are given in Lst. 1 from the main text. Prop. A.1 below establishes that distributions specified by DistInt and DistReal (Lst. 1e) with CDF F can be sampled using a variant of the integral probability transform.

Proposition A.1. *Let F be a* CDF *and r_1, r_2 real numbers such that $F(r_1) < F(r_2)$. Let $U \sim \text{Uniform}(F(r_1), F(r_2))$ and define the random variable $X := F^{-1}(U)$. Then for all real numbers r,*

$$\tilde{F}(r) := \Pr[X \le r] = \begin{cases} 0 & \text{if } r < r_1 \\ \dfrac{F(r) - F(r_1)}{F(r_2) - F(r_1)} & \text{if } r_1 \le r \le r_2 \\ 1 & \text{if } r_2 < r \end{cases} \tag{8}$$

Proof. Immediate from $\Pr[X \le r] = \Pr[U \le F(r)]$ and the uniformity of U on $[r_1, r_2]$. ◻

B Definitions of Auxiliary Functions

Sec. 3 refers to the following operations on the Outcomes domain:

$$\mathit{union} : \text{Outcomes}^* \to \text{Outcomes}, \tag{9}$$
$$\mathit{intersection} : \text{Outcomes}^* \to \text{Outcomes}, \tag{10}$$
$$\mathit{complement} : \text{Outcomes} \to \text{Outcomes}. \tag{11}$$

Any implementation satisfies the following invariants:

$$v_1 \amalg \cdots \amalg v_m = \mathit{union}\, v^* \iff \forall i \ne j. \mathit{intersection}\, v_i\, v_j = \varnothing, \tag{12}$$
$$v_1 \amalg \cdots \amalg v_m = \mathit{intersection}\, v^* \iff \forall i \ne j. \mathit{intersection}\, v_i\, v_j = \varnothing, \tag{13}$$
$$v_1 \amalg \cdots \amalg v_m = \mathit{complement}\, v \iff \forall i \ne j. \mathit{intersection}\, v_i\, v_j = \varnothing. \tag{14}$$

Lst. 10 shows an implementation of the *complement* function, which operates separately on the Real and String components; *union* and *intersection* are implemented similarly. Lst. 11 shows the *vars* function, which returns the variables in a Transform or Event expression. Lst. 14 shows the *negate* function, which returns the logical negation of an Event.

C Transforms of Random Variables

This appendix describes the Transform domain in the core calculus (expanding Lst. 1b), which is used to express numerical transformations of real random variables.

C.1 Valuation of Transforms

Lst. 17 shows the valuation function $\mathbb{T}$ which defines each t as a Real function on Real. Each real function $[\![T]\!]\, t$ is defined on an input r' if and only if $\left(\downarrow^{\text{Real}}_{\text{Outcome}} r'\right) \in (\mathit{domainof}\ t)$. Lst. 18 shows the implementation of *domainof*.

C.2 Preimage Computation

Lst. 19 shows the algorithm that implements

$$\mathit{preimg} : \text{Transform} \to \text{Outcomes} \to \text{Outcomes}, \tag{15}$$

which, as discussed in Sec. 3 of the main text, satisfies

$$(\downarrow^{\text{Real}}_{\text{Outcome}} r) \in \mathbb{V}[\![\mathit{preimg}\ t\ v]\!] \iff \mathbb{T}[\![t]\!](r) \in \mathbb{V}[\![v]\!],$$
$$(\downarrow^{\text{String}}_{\text{Outcome}} s) \in \mathbb{V}[\![\mathit{preimg}\ t\ v]\!] \iff (t \in \text{Identity}) \land (s \in \mathbb{V}[\![v]\!]).$$

The implementation of *preimg* uses several helper functions:

(Lst. 20) *finv*: computes the preimage of each $t \in$ Transform at a single Real.
(Lst. 21) *polyLim*: computes the limits of a polynomial at the infinites.
(Lst. 22) *polySolve*: computes the set of values at which a polynomial is equal to a given value (possibly positive or negative infinity).
(Lst. 23) *polyLte*: computes the set of values at which a polynomial is less than or equal a given value.

In addition, we assume access to a general root finding algorithm $\mathit{roots} : \text{Real}^+ \to \text{Real}^*$ (not shown), that returns a (possibly empty) list of roots of the degree-m polynomial with specified coefficients. In the reference implementation of Sppl, the *roots* function uses symbolic analysis for polynomials whose degree is less than or equal to two and semi-symbolic analysis for higher-order polynomials.

C.3 Example of Exact Inference on a Many-to-One Random Variable Transformation

This appendix shows how Sppl enables exact inference on many-to-one transformations of real random variables described in the previous section, where the transformation is itself determined by a stochastic branch (Fig. 4 in main text). Fig. 4a shows an Sppl program that defines a pair of random variables (X, Z), where X is normally distributed; and $Z = -X^3 + X^2 + 6X$ if $X < 1$, otherwise $Z = 5\sqrt{X} + 1$. The first plot of Fig. 4e shows the prior distribution of X; the middle plot shows the transformation t that defines $Z = t(X)$, which is a piecewise sum of t_{if} and t_{else}; and the final plot shows the distribution of $Z = t(X)$. Fig. 4b shows the sum-product expression representing this program, where the root node is a sum whose left and right children have weights 0.691... and 0.309..., which corresponds to the prior probabilities of $\{X < 1\}$ and $\{1 \le X\}$. Nodes labeled $X \sim N(\mu, \sigma)$ with an incoming directed edge from a node labeled (r_1, r_2) denotes that the random variable is constrained to the interval (r_1, r_2) (and similarly for closed intervals). Deterministic transformations are denoted by using red directed edges from a leaf node (i.e., X) to a numeric expression (e.g., $5\sqrt{X} + 11$), with the name of the transformed variable along the edge (i.e., Z).

Fig. 4c shows an Sppl query that conditions the program on an event $\{Z^2 \le 4\} \cap \{Z \ge 0\}$ involving the transformed variable Z. The inference engine performs the following analysis on the query:

$$
\begin{aligned}
&\{Z^2 \le 4\} \cap \{Z \ge 0\} && (16)\\
&\equiv \{Z \in [0,2]\} && (17)\\
&\equiv \{X \in t^{-1}([0,2])\} \qquad \text{recall } (Z \coloneqq t(X)) && (18)\\
&\equiv \{X \in t_{\text{if}}^{-1}([0,2])\} \cup \{X \in t_{\text{else}}^{-1}([0,2])\} && (19)\\
&\equiv \underbrace{\{-2.174\ldots \le X \le -2\} \cup \{0 \le X \le .321\ldots\}}_{\text{constraints from left subtree}} && (20)\\
&\qquad \cup \underbrace{\{81/25 \le X \le 121/25\}}_{\text{constraint from right subtree}}
\end{aligned}
$$

Eq. (17) shows the first stage of inference, which solves any transformations in the conditioning event and yields $\{0 \le Z \le 2\}$. The conditional distribution of Z is shown in the final plot of Fig. 4f. The next step is to dispatch the simplified event to the left and right subtrees. Each subtree will compute the constraint on X implied by the event under the transformation in that branch, as shown in Eq. (19). The middle plot of Fig. (4f) shows the preimage computation under t_{if} from the left subtree, which gives two intervals, and t_{else} from the right subtree, which gives one interval.

The final step is to transform the prior expression (Fig. 4b) by conditioning each subtree on the intervals in Eq. (20), which gives the posterior expression (Fig. 4d). The left subtree in Fig. 4b, which originally corresponded to $\{X < 1\}$, is split in Fig. 4d into two subtrees that represent the events $\{-2.174\ldots \le X \le -2\}$ and $\{0 \le X \le 0.321\ldots\}$, respectively, and whose weights 0.159... and 0.494... are the (renormalized) probabilities of these regions under the prior distribution (first plot of Fig. 4e). The right subtree in Fig. 4b, which originally corresponded to $\{1 \le X\}$, is now restricted to $\{81/25 \le X \le 121/25\}$ in Fig. 4d and its weight 0.347... is again the (renormalized) prior probability of the region. The graph in Fig. 4d represents the distribution of (X, Z) conditioned on the query in Eq. (17). The new sum-product expression be used to run further queries, such as using **simulate** to generate n i.i.d. random samples $\{(x_i, z_i)\}_{i=1}^n$ from the posterior distributions in Fig. 4f or **condition** to condition the program on further events.

D Conditioning Sum-Product Expressions

This section presents algorithms for exact inference, that is, conditioning the distribution defined by an element of SPE (Lst. 1f). Sec. D.2 focuses on a positive probability Event (Lst. 1c) and Sec. D.3 focuses on a Conjunction of equality constraints on non-transformed variables, such as $\{X = 3\} \cap \{Y = 4\}$ (see also Remark 4.2 in the main text). We will first prove Thm. 4.1 from the main text, which establishes that SPE is closed under conditioning on any positive probability Event. For completeness, we restate the Thm. 4.1 below.

Theorem 4.1 (Closure under conditioning). *Let $S \in$ SPE and $e \in$ Event be given, where $\mathbb{P}[\![S]\!]\, e > 0$. There exists an algorithm which, given S and e, returns $S' \in$ SPE such that, for all $e' \in$ Event, the probability of e' according to S' is equal to the conditional probability of e' given e according to S, i.e.,*

$$\mathbb{P}[\![S']\!]\, e' \equiv \mathbb{P}[\![S]\!]\,(e' \mid e) \coloneqq \frac{\mathbb{P}[\![S]\!]\,(e \sqcap e')}{\mathbb{P}[\![S]\!]\, e}. \qquad (5)$$

Thm. 4.1 is a structural conjugacy property [20] for the family of probability distributions defined by the SPE domain, where both the prior and posterior are identified by elements of SPE. In Sec. D.2, we present the domain function *condition* (Eq. (6), main text) which proves Thm. 4.1 by construction. We first discuss several preprocessing algorithms that are key subroutines used by *condition.*

D.1 Algorithms for Event Preprocessing

Normalizing an Event The *dnf* function (Lst. 15) converts an Event e to DNF, which we define below.

Definition D.1. An Event e is said to be in disjunctive normal form (DNF) if and only if one of the following holds:

(D.1.1) $e \in$ Containment
(D.1.2) $e = e_1 \sqcap \cdots \sqcap e_m \in$ Conjunction
$\qquad \implies \forall_{1 \le i \le m}.\ e_i \in$ Containment
(D.1.3) $e = e_1 \sqcup \cdots \sqcup e_m \in$ Disjunction
$\qquad \implies \forall_{1 \le i \le m}.\ e_i \in$ Containment $\cup$ Conjunction

Terms e and e_i in (D.1.1) and (D.1.2) are called "literals" and terms e_i in (D.1.3) are called "clauses".

We next define the notion of an Event in "solved" DNF.

Definition D.2. An Event e is in solved DNF if all the following conditions hold: (i) e is in DNF; (ii) all literals within a clause e_i of e have different variables; and (iii) each literal $(t \,\text{in}\, v)$ of e satisfies $t \in$ Identity and $v \notin$ Union.

Example D.3. Using informal notation, the solved DNF form of the event $\{X^2 \ge 9\} \cap \{|Y| < 1\}$ is a disjunction with two conjunctive clauses: $[\{X \in (-\infty, -3)\} \cap \{Y \in (-1,1)\}] \cup [\{X \in (3,\infty)\} \cap \{Y \in (-1,1)\}]$.

Lst. 5a shows the *normalize* operation, which converts an Event e to solved DNF. In particular, predicates with (possibly nonlinear) arithmetic expressions are converted to predicates that contain only linear expressions (which is a property of Transform and *preimg*; Appx. C); e.g., as in Eqs. (17)–(20). The next result, Prop. D.4, follows from $\mathbb{E}[\![e]\!] = \mathbb{E}[\![\mathit{dnf}\ e]\!]$ and denotations of Union (Lst. 1a) and Disjunction (Lst. 1c).

Proposition D.4. $\forall e \in$ Event, $\mathbb{E}[\![e]\!] \equiv \mathbb{E}[\![(\mathit{normalize}\, e)]\!]$.

Disjoining an Event Suppose that $e \in$ Event is in DNF and has $m \ge 2$ clauses. A key inference subroutine is to rewrite e in solved DNF (Def. D.2) where all the clauses are disjoint.

Definition D.5. Let $e \in$ Event be in DNF. Two clauses e_i and e_j of e are said to be disjoint if both e_i and e_j are in solved DNF and at least one of the following conditions holds:

$$\exists x \in (vars\, e_i).\ \mathbb{E}\,[\![e_{ix}]\!]\, x \equiv \varnothing \tag{21}$$

$$\exists x \in (vars\, e_j).\ \mathbb{E}\,[\![e_{jx}]\!]\, x \equiv \varnothing \tag{22}$$

$$\exists x \in (vars\, e_i) \cap (vars\, e_j).\ \mathbb{E}\,[\![e_{ix} \sqcap e_{jx}]\!]\, x \equiv \varnothing \tag{23}$$

where e_{ix} denotes the unique literal of e_i that contains variable x (for each $x \in vars\, e_i$), and similarly for e_j.

Lst. 16 shows the *disjoint?* procedure, which given a pair of clauses e_i and e_j that are in solved DNF (as produced by *normalize*), returns true if and only if one of the conditions in Def. D.5 hold. Lst. 5b presents the main algorithm *disjoin*, which decomposes an arbitrary Event e into solved DNF whose clauses are mutually disjoint. Prop. D.6 establishes the correctness and worst-case complexity of *disjoin*.

Proposition D.6. *Let e be an* Event *with $h := |vars\, e|$ variables, and suppose that $e_1 \sqcup \cdots \sqcup e_m := (normalize\, e)$ has exactly $m \geq 1$ clauses. Put $\tilde{e} := (disjoin\, e)$. Then:*

(D.6.1) $\tilde{e}$ is in solved DNF.
(D.6.2) $\forall_{1 \leq i \neq j \leq \ell}$. disjoint? $\langle e_i, e_j \rangle$.
(D.6.3) $\mathbb{E}\,[\![e]\!] = \mathbb{E}\,[\![\tilde{e}]\!]$.
(D.6.4) The number ℓ of clauses in $\tilde{e}$ satisfies $\ell \leq (2m-1)^h$.

Proof. Suppose first that $(normalize\, e)$ has $m = 1$ clause e_1. Then $\tilde{e} = e_1$, so (D.6.1) holds since $e_1 = normalize\, e$; (D.6.2) holds trivially; (D.6.3) holds by Prop. D.4; and (D.6.4) holds since $\ell = (2-1)^h = 1$. Suppose now that $(normalize\, e)$ has $m > 1$ clauses. To employ set-theoretic reasoning, fix some $x \in$ Var and define $\mathbb{E}'\,[\![e]\!] := \mathbb{V}\,[\![\mathbb{E}\,[\![e]\!]\, x]\!] \subset$ Outcome, for all $e \in$ Event. We have

$$\mathbb{E}'\,[\![e_1 \sqcup \cdots \sqcup e_m]\!] \tag{25}$$

$$= \cup_{i=1}^{m} \mathbb{E}'\,[\![e_i]\!] \tag{26}$$

$$= \cup_{i=1}^{m} \left(\mathbb{E}'\,[\![e_i]\!] \cap \neg \left[\cup_{j=1}^{i-1} (\mathbb{E}'\,[\![e_j]\!])\right]\right) \tag{27}$$

$$= \cup_{i=1}^{m} \left(\mathbb{E}'\,[\![e_i]\!] \cap \left[\cap_{j=1}^{i-1} (\neg \mathbb{E}'\,[\![e_j]\!])\right]\right) \tag{28}$$

$$= \cup_{i=1}^{m} \left(\mathbb{E}'\,[\![e_i]\!] \cap \left[\cap_{j \in k(i)} (\neg \mathbb{E}'\,[\![e_j]\!])\right]\right) \tag{29}$$

where we define for each $i = 1, \dots, m$,

$$k(i) := \left\{1 \leq j \leq i-1 \mid \mathbb{E}'\,[\![e_i]\!] \cap \mathbb{E}'\,[\![e_j]\!] \neq \varnothing\right\}.$$

Eq. (29) follows from the fact that for any $i = 1, \dots, m$ and $j < i$, we have

$$j \notin k(i) \implies \left[(\mathbb{E}'\,[\![e_i]\!] \cap \neg \mathbb{E}'\,[\![e_j]\!]) \equiv \mathbb{E}'\,[\![e_i]\!]\right]. \tag{30}$$

As *negate* (Lst. 14) computes set-theoretic complement $\neg$ in the Event domain and $j \notin k(i)$ if and only if $(disjoint?\ e_j\ e_i)$, it follows that the Events $e'_i := e_i \sqcap \tilde{e}_i$ $(i = 2, \dots, m)$ in Eq. (24c) are pairwise disjoint and are also disjoint from e_1, so that $\mathbb{E}\,[\![e]\!] = \mathbb{E}\,[\![e_1 \sqcup e'_2 \sqcup \cdots \sqcup e'_m]\!]$. Thus, if *disjoin* halts, then all of (D.6.1)–(D.6.3) follow by induction.

We next establish that *disjoin* halts by upper bounding the number of clauses ℓ returned by any call to *disjoin*. Recalling that $h := |vars\, e|$, we assume without loss of generality that all clauses e_i $(i = 1, \dots, n)$ in Eq. (24a) have the same variables $\{x_1, \dots, x_h\}$, by "padding" each e_i with vacuously true literals of the form `(Id(`x_i`) in Outcomes)`. Next, recall that clause e_i in Eq. (24a) is in solved DNF and has $m_i \geq 1$ literals e_{ij} = `(Id(`x_{ij}`) in` v_{ij}`)` where $v_{ij} \notin$ Union (Def. D.2). Thus, e_i specifies exactly one hyperrectangle in h-dimensional space, where v_{ij} is the "interval" (possibly infinite) along the dimension specified by x_{ij} in literal e_{ij} $(i = 1, \dots, m; j = 1, \dots, m_i)$. A sufficient condition to produce the worst-case number of pairwise disjoint primitive sub-hyperrectangles that partition the region $e_1 \sqcup \cdots \sqcup e_m$ is when the previous clauses $e_1, \dots, e_{m-1}$ (i) are pairwise disjoint (Def. D.5); and (ii) are strictly contained in e_m, i.e., $\forall x.\ \mathbb{E}\,[\![e_j]\!] \subsetneq \mathbb{E}\,[\![e_m]\!]$, $(j = 1, \dots, m-1)$. If these two conditions hold, then *disjoin* partitions the interior of the h-dimensional hyperrectangle specified by e_m into no more than $2(m-1)^h$ sub-hyperrectangles that do not intersect one another (and thus, produce no further recursive calls), thereby establishing (D.6.4). ◻

Example D.7. The left panel in Fig. 9 shows $m = 4$ rectangles in Real × Real. The right panel shows a grid (in red) with $(2m-1)^2 = 49$ primitive rectangular regions that are pairwise disjoint from one another and whose union over-approximates the union of the 4 rectangles. In this case, 29 of these primitive rectangular regions are sufficient (but excessive) to exactly partition the union of the rectangles into a disjoint union. No more than 49 primitive rectangles are ever needed to partition any 4 rectangles in Real^2, and this bound is tight. The bound in (D.6.4) generalizes this idea to hyperrectangles that live in h-dimensional space.

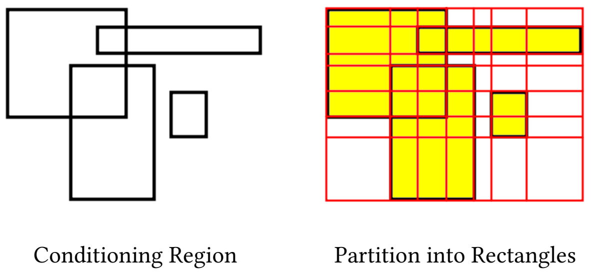

Figure 9. Example illustrating the upper bound (D.6.4) on the number of disjoint rectangles in a worst-case partition of a conditioning region in the two-dimensional Real plane.

Remark D.8. When defining $\tilde{e}$ in Eq (24b) of *disjoin*, ignoring previous clauses that are disjoint from e_i is essential for *disjoin* to halt, so as to avoid recursing on a primitive sub-rectangle in the interior. That is, filtering out such clauses ensures that *disjoin* makes a finite number of recursive calls.

```
$normalize$ : Event → Event
$normalize\ (t \text{ in } v) := \textbf{match}\ preimg\ t\ v$
  $\triangleright\ v'_1 \amalg \cdots \amalg v'_m \Rightarrow \sqcup_{i=1}^{m} (\mathtt{Id}(x) \text{ in } v'_i)$
  $\triangleright\ v' \Rightarrow (\mathtt{Id}(x) \text{ in } v'),$      where $\{x\} := vars\ t$
$normalize\ (e_1 \sqcap \cdots \sqcap e_m) := dnf\ \sqcap_{i=1}^{m} (normalize\ e_i)$
$normalize\ (e_1 \sqcup \cdots \sqcup e_m) := dnf\ \sqcup_{i=1}^{m} (normalize\ e_i)$
```

(a) normalize

```
$disjoin$ : Event → Event
$disjoin\ e := \textbf{let}\ (e_1 \sqcup \cdots \sqcup e_m)\ \textbf{be}\ normalize\ e$    (24a)
  $\textbf{in let}_{2 \le i \le m}\ \tilde{e}\ \textbf{be} \bigsqcap_{1 \le j < i \,\mid\, \neg(disjoint?\langle e_j, e_i \rangle)} (negate\ e_j)$    (24b)
  $\textbf{in let}_{2 \le i \le m}\ \tilde{e}_i\ \textbf{be}\ (disjoin\,(e_i \sqcap \tilde{e}_i))$    (24c)
  $\textbf{in}\ e_1 \sqcup \tilde{e}_2 \sqcup \cdots \sqcup \tilde{e}_m$
```

(b) disjoin

Listing 5. Event preprocessing algorithms used by *condition*.

```
$condition\ \mathtt{Leaf}(x\ d\ \sigma)\ e\ := \textbf{let}\ v\ \textbf{be}\ \mathbb{E}\,[\![(subsenv\ e\ \sigma)]\!]\ x\ \textbf{in match}\ d$
 $\triangleright\ \mathtt{DistS}((s_i\ w_i)_{i=1}^{m}) \Rightarrow \textbf{match}\ v$
   $\triangleright\ \{s'_1 \ldots s'_l\}^{b} \Rightarrow \textbf{let}_{1 \le i \le m}\ w'_i\ \textbf{be if}\ \bar{b}\ \textbf{then}\ w_i \mathbf{1}[\exists_{1 \le j \le \ell} . s'_j = s_i]$
                                        $\textbf{else}\ w_i \mathbf{1}[\forall_{1 \le j \le \ell} . s'_j \ne s_i]$
                         $\textbf{in}\ \mathtt{Leaf}(x\ \mathtt{DistS}((s_i\ w'_i)_{i=1}^{m})\ \sigma)$
   $\triangleright$ else undefined
 $\triangleright\ \mathtt{DistR}(F\ r_1\ r_2) \Rightarrow \textbf{match}\ (intersection\ ((\#f\ r_1)\ (r_2\ \#f))\ v)$
   $\triangleright\ \varnothing \mid \{r_1\ \ldots\ r_m\} \Rightarrow$ undefined
   $\triangleright\ ((b_1\ r'_1)\ (r'_2\ b_2)) \Rightarrow \mathtt{Leaf}(x\ \mathtt{DistR}(F\ r'_1\ r'_2)\ \sigma)$
   $\triangleright\ v_1 \amalg \cdots \amalg v_m \Rightarrow \textbf{let}_{1 \le i \le m}\ w_i\ \textbf{be}\ \mathbb{D}\,[\![d]\!]\ v_i$
     $\textbf{in let}\ \{n_1, \ldots, n_k\}\ \textbf{be}\ \{n \mid 0 < w_n\}$
     $\textbf{in let}_{1 \le i \le k}\ S_i\ \textbf{be}\ (condition\ \mathtt{Leaf}(x\ d\ \sigma)\ (\mathtt{Id}(x) \text{ in } v_{n_i}))$
     $\textbf{in if}\ (k = 1)\ \textbf{then}\ S_1\ \textbf{else}\ \oplus_{i=1}^{k} (S'_i\ w_{n_i})$
 $\triangleright\ \mathtt{DistI}(F\ r_1\ r_2) \Rightarrow \textbf{match}\ (intersection\ ((\#f\ r_1)\ (r_2\ \#f))\ v)$
   $\triangleright\ \{r_1\ \ldots\ r_m\} \Rightarrow \textbf{let}_{1 \le i \le m}\ w_i\ \textbf{be}\ \mathbb{D}\,[\![d]\!]\,\{r_i\}$
     $\textbf{in let}\ \{n_1, \ldots, n_k\}\ \textbf{be}\ \{n \mid 0 < w_n\}$
     $\textbf{in let}_{1 \le i \le k}\ S_i = (x\ \mathtt{DistI}(F\ (r_{n_i} - 1/2)\ r_{n_i})\ \sigma)$
     $\textbf{in if}\ (k = 1)\ \textbf{then}\ S_1\ \textbf{else}\ \oplus_{i=1}^{k} (S'_i\ w_{n_i})$
   $\triangleright$ else // same as last two cases forDistR
```

(a) Conditioning Leaf

```
$condition\ ((S_1\ w_1) \oplus \cdots \oplus (S_m\ w_m))\ e :=$
 $\textbf{let}_{1 \le i \le m}\ w'_i\ \textbf{be}\ w_i\ (\mathbb{P}\,[\![S_i]\!]\ e)$
 $\textbf{in let}\ \{n_1, \ldots, n_k\}\ \textbf{be}\ \{n \mid 0 < w'_n\}$
 $\textbf{in let}_{1 \le i \le k}\ S'_i\ \textbf{be}\ (condition\ S_{n_i}\ e)$
 $\textbf{in if}\ (k = 1)\ \textbf{then}\ S'_1\ \textbf{else}\ \oplus_{i=1}^{k} (S'_i\ w'_{n_i})$
```

(b) Conditioning Sum

```
$condition\ (S_1 \otimes \cdots \otimes S_m)\ e :=$
$\textbf{match}\ disjoin\ e$
 $\triangleright\ e_1 \sqcap \cdots \sqcap e_h \Rightarrow$ //one $h$-dimensional hyperrectangle
   $\bigotimes_{1 \le i \le m} \left[\begin{array}{l} \textbf{match}\ \{1 \le j \le h \mid (vars\ e_j) \subset (scope\ S_i)\} \\ \triangleright\ \{n_1, \ldots, n_k\} \\ \quad \Rightarrow condition\ S_i\ (e_{n_1} \sqcap \cdots \sqcap e_{n_k}) \\ \triangleright\ \{\} \Rightarrow S_i \end{array}\right]$
 $\triangleright\ e_1 \sqcup \cdots \sqcup e_\ell \Rightarrow$ //$\ell \ge 2$ disjoint hyperrectangles
   $\textbf{let}_{1 \le i \le \ell}\ w_i\ \textbf{be}\ \mathbb{P}\,[\![S_1 \otimes \cdots \otimes S_m]\!]\ e_i$
   $\textbf{in let}\ \{n_1, \ldots, n_k\}\ \textbf{be}\ \{n \mid 0 < w_n\}$
   $\textbf{in let}_{1 \le i \le k}\ S'_i\ \textbf{be}\ (condition\,(S_1 \otimes \cdots \otimes S_m)\ e_{n_i})$
   $\textbf{in if}\ (k = 1)\ \textbf{then}\ S'_1\ \textbf{else}\ \oplus_{i=1}^{k} (S'_i\ w_{n_i})$
```

(c) Conditioning Product

Listing 6. Implementation of *condition* for Leaf, Sum, and Product expressions using distribution semantics in Lst. 1e.

D.2 Algorithms for Conditioning Sum-Product Expressions on Positive Measure Events

Having established the key background details, we now prove Thm. 4.1 from the main text, which establishes the closure under conditioning property of the SP domain.

Proof of Theorem. 4.1. We establish Eq. (5) by defining

$$condition : \mathsf{SPE} \longrightarrow \mathsf{Event} \longrightarrow \mathsf{SPE} \tag{31}$$

which satisfies

$$\mathbb{P}\,[\![(condition\,S\,e)]\!]\ e' = \frac{\mathbb{P}\,[\![S]\!]\,(e \sqcap e')}{\mathbb{P}\,[\![S]\!]\ e} \tag{32}$$

for all $e' \in$ Event and $e \in$ Event for which $\mathbb{P}\,[\![S]\!]\ e > 0$.

We will define *condition* separately for each of the three constructors Leaf, Sum, and Product from Lst. 9f. The proof is by structural induction, where Leaf is the base case and Sum and Product are the recursive cases.

Conditioning Leaf Lst. 6a shows the base cases of *condition*. The case of $d \in$ DistStr is straightforward. For $d \in$ DistReal, if the intersection (defined in second line of Lst. 6a) of v with the support of d is an interval $((b'_1\ r'_1)\ (r'_2, b'_2))$, then it suffices to return a Leaf restricting d to the interval. If the intersection is a Union $v_1 \amalg \cdots \amalg v_m$ (recall fro Eq. (13) that *intersection* ensures the v_i are disjoint), then the conditioned SPE is a Sum, whose ith child is obtained by recursively calling *condition* on v_i and ith (relative) weight is the probability of v under d, since, for any new $v' \in$ Outcomes, we have

$$\begin{aligned} &\frac{\mathbb{D}\,[\![d]\!]\,(intersect\,v'\,(v_1 \amalg \cdots \amalg v_m))}{\mathbb{D}\,[\![d]\!]\,(v_1 \amalg \cdots \amalg v_m)} \\ &\quad = \frac{\mathbb{D}\,[\![d]\!]\,\amalg_{i=1}^{m}(intersect\,v'\,v_i)}{\sum_{i=1}^{m} \mathbb{D}\,[\![d]\!]\,v_i}. \end{aligned} \tag{33}$$

Eq. (33) follows from the additivity of $\mathbb{D}\,[\![d]\!]$. The plots of X in Figs. 4e and 4f illustrate the identity in Eq. (33), where conditioning the unimodal normal distribution results in a mixture of three restricted normals whose weights are given by the relative prior probabilities of the three regions.

For $d \in$ DistInt, if the positive probability Outcomes are $\{r_1 \ldots r_m\}$, then the conditioned SPE is a Sum of "delta"-CDFs whose atoms are located on the integers r_i and weights are the (relative) probabilities $\mathbb{D}\,[\![d]\!]\,\{r_i\}\ (i = 1, \ldots, m)$. Since the atoms of F for DistInt are integers, it suffices to restrict F to the interval $(r_i - 1/2, r_i)$, for each r_i with a positive weight. Correctness again follows from Eq. (33), since finite sets are unions of disjoint singleton sets. For other positive probability Outcomes, the conditioning procedure DistInt is the same as that for DistReal.

Conditioning Sum Lst. 6b shows *condition* for $S \in$ Sum. Recalling the denotation $\mathbb{P}[\![S]\!]$ for $S \in$ Sum in Lst. 1f, the correctness follows from the following properties:

$$\frac{\mathbb{P}[\![(S_1\ w_1) \oplus \cdots \oplus (S_m\ w_m)]\!]\,(e \sqcap e')}{\mathbb{P}[\![(S_1\ w_1) \oplus \cdots \oplus (S_m\ w_m)]\!]\,e} \tag{34}$$

$$= \frac{\sum_{i=1}^{m} w_i \mathbb{P}[\![S_i]\!]\,(e \sqcap e')}{\sum_{i=1}^{m} w_i \mathbb{P}[\![S_i]\!]\,e} \tag{35}$$

$$= \frac{\sum_{i=1}^{m} w_i (\mathbb{P}[\![S_i]\!]\,e) \mathbb{P}[\![(\mathit{condition}\,S_i\,e)]\!]\,e'}{\sum_{i=1}^{m} w_i \mathbb{P}[\![S_i]\!]\,e} \tag{36}$$

$$= \mathbb{P}[\![\oplus_{i=1}^{m}((\mathit{condition}\,S_i\,e)\,, w_i \mathbb{P}[\![S_i]\!]\,e)]\!]\,e', \tag{37}$$

where Eq. (36) has applied Eq. (32) inductively for each S_i. Eqs. (35)–(36) assume for simplicity that $\mathbb{P}[\![S_i]\!]\,e > 0$ for each $i = 1, \dots, m$, whereas Lst. 6a does not make this assumption.

Conditioning Product Lst. 6c how *condition* operates on $S \in$ Product. The first step is to invoke *disjoin* to rewrite $(\mathit{dnf}\ e)$ as $\ell \geq 1$ disjoint clauses $e'_1 \sqcup \cdots \sqcup e'_\ell$ (recall from Prop. D.6 that *disjoin* is semantics-preserving). The first pattern in the **match** statement corresponds $\ell = 1$, and the result is a new Product, where the ith child is conditioned on the literals of e_1 whose variables are contained in $\mathit{scope}\,S_i$ (if any). The second pattern returns a Sum of Product, since

$$\frac{\mathbb{P}[\![S_1 \otimes \cdots \otimes S_m]\!]\,(e \sqcap e')}{\mathbb{P}[\![S_1 \otimes \cdots \otimes S_m]\!]\,e} \tag{38}$$

$$= \frac{\mathbb{P}[\![S_1 \otimes \cdots \otimes S_m]\!]\,((e_1 \sqcup \cdots \sqcup e_\ell) \sqcap e')}{\mathbb{P}[\![S_1 \otimes \cdots \otimes S_m]\!]\,(e_1 \sqcup \cdots \sqcup e_\ell)} \tag{39}$$

$$= \frac{\mathbb{P}[\![S_1 \otimes \cdots \otimes S_m]\!]\,((e_1 \sqcap e') \sqcup \cdots \sqcup (e_\ell \sqcap e'))}{\sum_{i=1}^{\ell} \mathbb{P}[\![S_1 \otimes \cdots \otimes S_m]\!]\,e_i} \tag{40}$$

$$= \frac{\sum_{i=1}^{\ell} \mathbb{P}[\![S_1 \otimes \cdots \otimes S_m]\!]\,(e_i \sqcap e')}{\sum_{i=1}^{\ell} \mathbb{P}[\![S_1 \otimes \cdots \otimes S_m]\!]\,e_i} \tag{41}$$

$$= \frac{\sum_{i=1}^{\ell} \mathbb{P}[\![S]\!]\,e_i\, \mathbb{P}[\![(\mathit{condition}\,(S_1 \otimes \cdots \otimes S_m)\,e_i)]\!]\,e'}{\sum_{i=1}^{\ell} \mathbb{P}[\![S_1 \otimes \cdots \otimes S_m]\!]\,e_i} \tag{42}$$

$$= \mathbb{P}\left[\!\!\left[\oplus_{i=1}^{\ell}((\mathit{condition}\,S\,e_i)\ \mathbb{P}[\![S]\!]\,e_i)\right]\!\!\right]\,e'. \tag{43}$$

Eq (42) follows from the induction hypothesis Eq. (32) and $(\mathit{disjoin}\,e_i) \equiv e_i$ (idempotence), so that $(\mathit{disjoin}\,e_i \sqcap e') \equiv (\mathit{disjoin}\,e_i) \sqcap (\mathit{disjoin}\,e') \equiv e_i \sqcap (\mathit{disjoin}\,e')$.

Thm. 4.1 is thus established. □

Fig. 5 in the main text shows an example of the closure property from Thm. 4.1, where conditioning on a hyperrectangle changes the structure of the SPE from a Product into a Sum-of-Product. The algorithms in this section are the first to describe probabilistic inference and closure properties for conditioning an SPE on a query that involves transforms of random variables and predicates with set-valued constraints. We next establish Thm. 4.3 from the main text, which gives a sufficient condition for the runtime of *condition* (Lst. 6) to scale linearly in the number of nodes in S; identical results hold for computing Event probabilities ($\mathbb{P}[\![S]\!]\,e$, Lst. 1f) and probability densities ($\mathbb{P}_0[\![S]\!]\,e$, Lst. 1d).

Theorem 4.3. *The runtime of* $(\mathit{condition}\,S\,e)$ *scales linearly in the number of nodes in the graph representing S whenever e is a single* Conjunction $(t_1 \,\mathtt{in}\, v_1) \sqcap \cdots \sqcap (t_m \,\mathtt{in}\, v_m)$ *of* Containment *constraints on non-transformed variables.*

Proof. First, if S is a Sum (Lst. 6b) with m children then $(\mathit{condition}\,S\,e)$ makes no more than m subcalls to *condition* (one for each child), and if S is a Leaf (Lst. 6a) then there are zero subcalls, independently of e. Since each node has exactly one parent, we can can conclude that each node in S is visited exactly once by showing that the hypothesis on e implies that for any $S \in$ Product (Lst. 6c) with m children, there are makes at most m subcalls to *condition* from which we (each node has exactly one parent). Suppose that $(\mathit{disjoin}\,e)$ returns a single Conjunction. Then the first pattern of the **match** statement in Lst. 6c is matched (one h-dimensional rectangle), resulting in m subcalls to *condition*. Thus, each node in S is visited (at most) once by *condition*. To complete the proof, note that the hypothesis that e specifies a single Conjunction $(t_1 \,\mathtt{in}\, v_1) \sqcap \cdots \sqcap (t_m \,\mathtt{in}\, v_m)$ of Containment constraints on non-transformed variables is sufficient for $(\mathit{disjoin}\,e)$ to return a single Conjunction. □

D.3 Conditioning Sum-Product Expressions on Measure Zero Equality Constraints

Recall from Remark 4.2 in the main text that SPE is also closed under conditioning on a Conjuction of possibly measure zero equality constraints of non-transformed variable, such as $\{X = 3, Y = \pi, Z = \texttt{"foo"}\}$. In this section, we describe the conditioning algorithm for this case, which is implemented by

$$\mathit{condition}_0 : \text{SPE} \to \text{Event} \to \text{SPE}, \tag{44}$$

where $e \in$ Event satisfies the follows requirements with respect to $S \in$ SPE:

1. Either $e \equiv (\mathtt{Id}(x)\ \mathtt{in}\,\{rs\})$ or e is a Conjunction of such literals, where $\equiv$ here denote syntactic (not semantic) equivalence.
2. Every Var x in each literal of e is a non-transformed variable; i.e., for each Leaf expression S such that $x \in \mathit{scope}\,S$, we have $S \equiv \mathtt{Leaf}(x\,d\,\sigma)$, for some d and σ.

With these requirements on e, Lst. 7 presents the implementation of $\mathit{condition}_0$, leveraging the generalized density semantics from Lst. 1d in the main text. The inference rules closely match those for standard sum-product networks, except for the fact that a density from $\mathbb{P}_0[\![S]\!]$ is a pair, whose first entry is the number of continuous distributions participating in the weight of the Event e which must be correctly accounted for by $\mathit{condition}_0$. In the reference implementation of Sppl, **`condition`** invokes *condition* and **`constrain`** invokes $\mathit{condition}_0$. Analogously to the **`prob`** query, which returns probabilities using the distribution semantics $\mathbb{P}$ in Lst. 9e, Sppl also includes the **`density`** query, which returns densities using the generalized semantics $\mathbb{P}_0$ in Lst. 1d.

```
$\mathit{condition}_0$ Leaf($x\ d\ \sigma$) (Id($x$) in {$rs$}) := match $d$
  ▷ DistR($F\ r_1\ r_2$) ⇒ match $rs$
    ▷ $r$ ⇒ match ($\mathbb{P}_0$ ⟦Leaf($x\ d\ \sigma$)⟧ (Id($x$) in {$rs$}))
      ▷ (1, 0) ⇒ undefined
      ▷ else let $\tilde{F}$ be $(\lambda r'.\, \mathbf{1}[r \le r'])$ in DistI($\tilde{F}\,(r - 1/2)\, r$)
    ▷ $s$ ⇒ undefined
  ▷ else ⇒ $\mathit{condition}$ Leaf($x\ d\ \sigma$) (Id($x$) in {$rs$})
```

(a) Conditioning Leaf

```
$\mathit{condition}_0\ ((S_1\ w_1) \oplus \cdots \oplus (S_m\ w_m))\ \left(\sqcap_{i=1}^{\ell}(\mathrm{Id}(x_i)\ \mathrm{in}\ \{rs_i\})\right)$ :=
let$_{1 \le i \le m}$ $(d_i, p_i)$ be $\mathbb{P}_0\, [\![S_i]\!]\, \left(\sqcap_{i=1}^{\ell}(\mathrm{Id}(x_i)\ \mathrm{in}\ \{rs_i\})\right)$
in if $\forall_{1 \le i \le m} .\ p_i = 0$ then undefined
  else let$_{1 \le i \le m}$ $w'_i$ be $w_i p_i$
    in let $d^*$ be $\min\{d_i \mid 1 \le i \le m, 0 < p_i\}$
    in let $\{n_1, \dots, n_k\}$ be $\{n \mid 0 < w'_n, d_i = d^*\}$
    in let$_{1 \le i \le k}$ $S'_i$ be $\left(\mathit{condition}_0\ S_{n_i}\ \left(\sqcap_{i=1}^{\ell}(\mathrm{Id}(x_i)\ \mathrm{in}\ \{rs_i\})\right)\right)$
    in if $(k = 1)$ then $S'_1$ else $\oplus_{i=1}^{k}\ (S'_i\ w_{n_i})$
```

(b) Conditioning Sum

```
$\mathit{condition}_0\ (S_1 \otimes \cdots \otimes S_m)\ \sqcap_{i=1}^{\ell}\ (\mathrm{Id}(x_i)\ \mathrm{in}\ \{rs_i\})$ :=
  let$_{1 \le i \le m}$ $S'_i$ be match $\{x_1, \dots, x_m\} \cap (\mathit{scope}\ S_i)$
    ▷ $\{n_1, \dots, n_k\}$ ⇒ $\mathit{condition}_0\ S_i\ \sqcap_{t=1}^{k}\ (\mathrm{Id}(x_{n_t})\ \mathrm{in}\ \{rs_t\})$
    ▷ {} ⇒ $S_i$
  in $S'_1 \otimes \cdots \otimes S'_m$
```

(c) Conditioning Product

Listing 7. Implementation of *condition*$_0$ for Leaf, Sum, and Product expressions using density semantics in Lst. 1d.

E Translating Sum-Product Expressions to Sppl Programs

Lst. 3 in Sec. 5 presents the relation $\to_{\mathrm{SPE}}$, that translates $C \in$ Command (i.e., Sppl source syntax, Lst. 2) to a sum-product expression $S \in$ SPE in the core language (Lst. 9). Lst. 8 defines a relation $\to_{\mathrm{Sppl}}$ that reverses the $\to_{\mathrm{SPE}}$ relation: it converts expression $S \in$ SPE to $C \in$ Command. Briefly, (i) a Product is converted to a sequence Command; (ii) a Sum is converted to an **if-else** Command; and (iii) a Leaf is converted to a sequence of sample (~) and transform (=). The symbol $\Uparrow$ (whose definition is omitted) in the (Leaf) rule converts semantic elements such as $d \in$ Distribution and $t \in$ Transform from the core calculus (Lst. 1) to an Sppl expression $E \in$ Expr (Lst. 2) in a straightforward way, e.g.,

$$\texttt{(Poly(Id(X) 1 2 3))} \Uparrow \texttt{(1 + 2*X + 3*X**2)}. \tag{45}$$

It is easy to see that chaining $\to_{\mathrm{SPE}}$ (Lst. 3) and $\to_{\mathrm{Sppl}}$ (Lst. 8) for a given Sppl program does not preserve either Sppl or core syntax, that is[3]

$$\begin{aligned} &((C \to^*_{\mathrm{SPE}} S) \to^*_{\mathrm{Sppl}} C') && \text{does not imply } C = C' \\ &((C \to^*_{\mathrm{SPE}} S) \to^*_{\mathrm{Sppl}} C') \to^*_{\mathrm{SPE}} S' && \text{does not imply } S = S'. \end{aligned}$$

[3]The symbol $C \to^*_{\mathrm{SPE}} S$ means $\langle C, S_\varnothing \rangle$ translates to S in zero or more steps of $\to_{\mathrm{SPE}}$, where $S_\varnothing$ is an "empty" SP used for the initial translation step, and similarly for $\to^*_{\mathrm{Sppl}}$.

(Leaf)

$$\frac{d \Uparrow D(E),\ t_1 \Uparrow E_1, \dots, t_m \Uparrow E_m}{(x\ d\ \{x \mapsto \mathrm{Id}(x), x_1 \mapsto t_1, \dots, x_m \mapsto t_m\}) \to_{\mathrm{Sppl}} x \sim D(E)\, ;\, x_1 = E_1\, ;\, \dots\, ;\, x_m = E_m}$$

(Product)

$$\frac{S_1 \to_{\mathrm{Sppl}} C_1, \dots, S_m \to_{\mathrm{Sppl}} C_m}{\otimes_{i=1}^{m} S_i \to_{\mathrm{Sppl}} C_1\, ;\, \dots\, ;\, C_m}$$

(Sum)

$$\frac{S_1 \to_{\mathrm{Sppl}} C_1, \dots, S_m \to_{\mathrm{Sppl}} C_m; \quad \text{where } b \text{ is a fresh Var}}{\oplus_{i=1}^{m} (S_i\ w_i) \to_{\mathrm{Sppl}} \begin{bmatrix} b \sim \texttt{choice}(\{\texttt{'1'}: w_1, \dots, \texttt{'}m\texttt{'}: w_m\}) \\ \textbf{if}\ (b == \texttt{'1'})\ \{C_1\} \\ \textbf{elif} \dots \\ \textbf{elif}\ (b == \texttt{'}m\texttt{'})\ \{C_m\} \end{bmatrix}}$$

Listing 8. Translating an element of SPE (Lst. 9f) to an Sppl command C (Lst. 2).

Instead, it can be shown that $\to_{\mathrm{Sppl}}$ is a semantics-preserving inverse of $\to_{\mathrm{SPE}}$, in the sense that for all $e \in$ Event

$$((C \to^*_{\mathrm{SPE}} S) \to^*_{\mathrm{Sppl}} C') \to^*_{\mathrm{SPE}} S' \implies \mathbb{P}\,[\![S]\!]\, e = \mathbb{P}\,[\![S']\!]\, e. \tag{46}$$

Eq. (46) establish a formal semantic correspondence between the Sppl language and the class of sum-product expressions: each Sppl program admits a representation as an SPE, and each valid element of SPE that satisfies conditions (C1)–(C5) expression corresponds to some Sppl program.

Thus, in addition to synthesizing full Sppl programs from data using the PPL synthesis systems [13, 53] mentioned in Sec. 7, it is also possible with the translation strategy in Lst. 8 to synthesize Sppl programs using the wide range of techniques for learning the structure and parameters of sum-product networks [26, 37, 62, 65]. With this approach, Sppl (i) provides users with a uniform representation of existing sum-product networks as generative source code in a formal PPL (Lst. 2); (ii) allows users to extend these baseline programs with modeling extensions supported by the core calculus (Lst. 1), such as predicates for decision trees and numeric transformations; and (iii) delivers exact answers to an extended set of probabilistic inference queries (Sec. 4) within the modular and reusable workflow from Fig. 1.

$$\begin{array}{l} x \in \text{Var} \\ n \in \text{Natural} \\ b \in \text{Boolean} := \{\#\mathtt{t}, \#\mathtt{f}\} \\ u \in \text{Unit} := \{\#\mathtt{u}\} \\ w \in [0, 1] \\ r \in \text{Real} \cup \{-\infty, \infty\} \\ s \in \text{String} := \text{Char}^* \end{array}$$

(a) Basic Sets

$$\begin{array}{ll} rs \in \text{Outcome} := \text{Real} + \text{String} & \\ v \in \text{Outcomes} & \\ \quad := \varnothing & [\text{Empty}] \\ \quad \mid \{s_1 \ldots s_m\}^b & [\text{FiniteStr}] \\ \quad \mid \{r_1 \ldots r_m\} & [\text{FiniteReal}] \\ \quad \mid ((b_1\, r_1)\ (r_2\, b_2)) & [\text{Interval}] \\ \quad \mid v_1 \amalg \cdots \amalg v_m & [\text{Union}] \end{array}$$

(b) Outcomes

$$\begin{array}{ll} t \in \text{Transform} & \\ \quad := \mathtt{Id}(x) & [\text{Identity}] \\ \quad \mid \mathtt{Reciprocal}(t) & [\text{Reciprocal}] \\ \quad \mid \mathtt{Abs}(t) & [\text{AbsValue}] \\ \quad \mid \mathtt{Root}(t\ n) & [\text{Radical}] \\ \quad \mid \mathtt{Exp}(t\ r) & [\text{Exponent}] \\ \quad \mid \mathtt{Log}(t\ r) & [\text{Logarithm}] \\ \quad \mid \mathtt{Poly}(t\ r_0 \ldots r_m) & [\text{Polynomial}] \\ \quad \mid \mathtt{Piecewise}((t_1\ e_1) & [\text{Piecewise}] \\ \qquad\qquad \ldots & \\ \qquad\qquad (t_m\ e_m)) & \end{array}$$

(c) Transformations

$$\begin{array}{ll} e \in \text{Event} & \\ \quad := (t\ \mathtt{in}\ v) & [\text{Containment}] \\ \quad \mid e_1 \sqcap \cdots \sqcap e_m & [\text{Conjunction}] \\ \quad \mid e_1 \sqcup \cdots \sqcup e_m & [\text{Disjunction}] \end{array}$$

(d) Events

$$\begin{array}{ll} F \in \text{CDF} \subset \text{Real} \to [0, 1] & \\ \quad := \mathtt{Norm}(r_1, r_2) \mid \mathtt{Poisson}(r) \mid \mathtt{Binom}(n, w) \ldots & \\ \quad \text{where } F \text{ is càdlàg;} & \\ \qquad \lim_{r \to \infty} F(r) = 1;\ \lim_{r \to -\infty} F(r) = 0; & \\ \qquad \text{and } F^{-1}(u) := \inf\{r \mid u \le F(r)\}. & \\ d \in \text{Distribution} & \\ \quad := \mathtt{DistR}(F\ r_1\ r_2) & [\text{DistReal}] \\ \quad \mid \mathtt{DistI}(F\ r_1\ r_2) & [\text{DistInt}] \\ \quad \mid \mathtt{DistS}((s_1\ w_1) \ldots (s_m\ w_m)) & [\text{DistStr}] \end{array}$$

(e) Primitive Distributions

$$\begin{array}{ll} \sigma \in \text{Environment} := \text{Var} \to \text{Transform} & \\ S \in \text{SPE} & \\ \quad := \mathtt{Leaf}(x\ d\ \sigma) & [\text{Leaf}] \\ \quad \mid (S_1\ w_1) \oplus \cdots \oplus (S_m\ w_m) & [\text{Sum}] \\ \quad \mid S_1 \otimes \cdots \otimes S_m & [\text{Product}] \end{array}$$

(f) Sum-Product

Listing 9. Core calculus.

$$\begin{aligned} \mathit{complement}\ \{s_1 \ldots s_m\}^b &:= \{s_1 \ldots s_m\}^{\neg b} \\ \mathit{complement}\ ((b_1\ r_1)\ (r_2\ b_2)) &:= ((\#\mathtt{f}\ {-\infty})\ (r_1\ \neg b_1)) \amalg ((\neg b_2\ r_2)\ (\infty\ \#\mathtt{f})) \\ \mathit{complement}\ \{r_1 \ldots r_m\} &:= ((\#\mathtt{f}\ {-\infty})\ (r_1\ \#\mathtt{t})) \\ &\qquad \amalg \left[\amalg_{j=2}^{m}((\#\mathtt{t}\ r_{j-1})\ (r_j\ \#\mathtt{t}))\right] \\ &\qquad \amalg ((\#\mathtt{t}\ r_m)\ (\infty\ \#\mathtt{f})) \\ \mathit{complement}\ \varnothing &:= \{\}^{\#\mathtt{t}} \amalg ((\#\mathtt{f}\ {-\infty})\ (\infty\ \#\mathtt{f})) \end{aligned}$$

Listing 10. Implementation of *complement* on the sum domain Outcomes.

```
$\mathit{vars} : (\text{Transform} + \text{Event}) \to \mathcal{P}(\text{Vars})$
$\mathit{vars}\ te = \textbf{match}\ te$
    $\triangleright\ t \Rightarrow \textbf{match}\ t$
        $\triangleright\ \mathtt{Id}(x) \Rightarrow \{x\}$
        $\triangleright\ \mathtt{Root}(t'\ n) \mid \mathtt{Exp}(t'\ r) \mid \mathtt{Log}(t'\ r) \mid \mathtt{Abs}(t')$
            $\mid \mathtt{Reciprocal}(t') \mid \mathtt{Poly}(t'\ r_0\ \ldots\ r_m)$
            $\Rightarrow \mathit{vars}\ t'$
        $\triangleright\ \mathtt{Piecewise}((t_i\ e_i)_{i=1}^{m}) \Rightarrow \cup_{i=1}^{m}((\mathit{vars}\ t_i) \cup (\mathit{vars}\ e_i))$
    $\triangleright\ (t\ \mathtt{in}\ v) \Rightarrow \mathit{vars}\ t$
    $\triangleright\ (e_1 \sqcap \cdots \sqcap e_m) \mid (e_1 \sqcup \cdots \sqcup e_m) \Rightarrow \cup_{i=1}^{m} \mathit{vars}\ e_i$
```

Listing 11. Implementation of *vars*, which returns the variables in a Transform or Event.

$$\begin{array}{l} \mathit{scope} : \text{SPE} \to \mathcal{P}(\text{Var}) \\ \mathit{scope}\ (x\ d\ \sigma) := \text{dom}(\sigma) \\ \mathit{scope}\ (S_1 \otimes \cdots \otimes S_m) := \cup_{i=1}^{m}(\mathit{scope}\ S_i) \\ \mathit{scope}\ ((S_1\ w_1) \oplus \cdots \oplus (S_m\ w_m)) := (\mathit{scope}\ S_1) \end{array}$$

Listing 12. Implementation of *scope*, which returns the set of variables in an element of SPE.

```
$subsenv : \mathsf{Event} \to \mathsf{Environment} \to \mathsf{Event}$
$subsenv\ e\ \sigma \coloneqq \textbf{let}\,\{x, x_1, \dots, x_m\} = \mathrm{dom}(\sigma)$
                  $\textbf{in}\,\textbf{let}\,e_1\ \textbf{be}\ subs\ e\ x_m\ \sigma(x_m)$
                  $\dots$
                  $\textbf{in}\,\textbf{let}\,e_m\ \textbf{be}\ subs\ e_{m-1}\ x_1\ \sigma(x_1)$
                  $\textbf{in}\,e_m$
```

Listing 13. Implementation of *subsenv*, which rewrites e as an Event e' on one variable x.

```
$negate : \mathsf{Event} \to \mathsf{Event}$
$negate\,(t\ \mathtt{in}\ v) \coloneqq \textbf{match}\ (complement\ v)$
    $\triangleright\ v_1 \amalg \cdots \amalg v_m \Rightarrow (t\ \mathtt{in}\ v_1) \sqcup \cdots \sqcup (t\ \mathtt{in}\ v_m)$
    $\triangleright\ v \Rightarrow (t\ \mathtt{in}\ v)$
$negate\,(e_1 \sqcap \cdots \sqcap e_m) \coloneqq \sqcup_{i=1}^{m}(negate\ e_i)$
$negate\,(e_1 \sqcup \cdots \sqcup e_m) \coloneqq \sqcap_{i=1}^{m}(negate\ e_i)$
```

Listing 14. Implementation of *negate*, which applies De Morgan's laws to an Event.

```
$dnf : \mathsf{Event} \to \mathsf{Event}$
$dnf\ (t\ \mathtt{in}\ v) \coloneqq (t\ \mathtt{in}\ v)$
$dnf\ e_1 \sqcup \cdots \sqcup e_m \coloneqq \sqcup_{i=1}^{m}(dnf\ e_i)$
$dnf\ e_1 \sqcap \cdots \sqcap e_m \coloneqq \textbf{let}_{1 \le i \le m}\ (e'_{j1} \sqcap \cdots \sqcap e'_{j,k_i})\ \textbf{be}\ dnf\ e_i$
              $\textbf{in} \bigsqcup_{\substack{1 \le j_1 \le k_1 \\ \dots \\ 1 \le j_m \le k_m}} \bigsqcap_{i=1}^{m} e'_{i,j_i}$
```

Listing 15. *dnf* converts and Event to DNF (Def. D.1).

```
$disjoint? : \mathsf{Event} \times \mathsf{Event} \to \mathsf{Boolean}$
$disjoint?\,\langle e_1, e_2 \rangle \coloneqq \textbf{match}\ \langle e_1, e_2 \rangle$
 $\triangleright\ \langle \sqcap_{i=1}^{m_1}(\mathtt{Id}(x_{1,i})\ \mathtt{in}\ v_{1,i}), \sqcap_{i=1}^{m_2}(\mathtt{Id}(x_{2,i})\ \mathtt{in}\ v_{2,i}) \rangle$
  $\Rightarrow \left[\exists_{1 \le i \le 2}.\exists_{1 \le j \le m_i}.v_{ij} = \varnothing)\right] \vee \begin{bmatrix}\textbf{let}\ \{\langle n_{1i}, n_{2i} \rangle\}_{i=1}^{k}\ \textbf{be}\ \{\langle i, j \rangle \mid x_{1,i} = x_{2,j}\} \\ \textbf{in}\ (\exists_{1 \le i \le k}.(intersection\, v_{1,n_{1,i}}\ v_{2,n_{2,i}}) = \varnothing)\end{bmatrix}$
 $\triangleright\ \textbf{else} \Rightarrow \textbf{undefined}$
```

Listing 16. *disjoint*? returns #t if two Events are disjoint (Def. D.5).

$$
\begin{aligned}
\mathbb{T} &: \text{Transform} \to (\text{Real} \to \text{Real}) \\
\mathbb{T}[\![\texttt{Id}(x)]\!] &\coloneqq \lambda r'.\, r' \\
\mathbb{T}[\![\texttt{Reciprocal}(t)]\!] &\coloneqq \lambda r'.\, 1/(\mathbb{T}[\![t]\!](r')) \\
\mathbb{T}[\![\texttt{Abs}(t)]\!] &\coloneqq \lambda r'.\, |\mathbb{T}[\![t]\!](r')| \\
\mathbb{T}[\![\texttt{Root}(t\ n)]\!] &\coloneqq \lambda r'.\, \sqrt[n]{\mathbb{T}[\![t]\!](r')} \\
\mathbb{T}[\![\texttt{Exp}(t\ r)]\!] &\coloneqq \lambda r'.\, r^{(\mathbb{T}[\![t]\!](r'))} && (\text{iff } 0 < r) \\
\mathbb{T}[\![\texttt{Log}(t\ r)]\!] &\coloneqq \lambda r'.\, \log_r(\mathbb{T}[\![t]\!](r')) && (\text{iff } 0 < r) \\
\mathbb{T}[\![\texttt{Poly}(t\ r_0\ \ldots\ r_m)]\!] &\coloneqq \lambda r'.\, \textstyle\sum_{i=0}^{m} r_i\,(\mathbb{T}[\![t]\!](r'))^i \\
\mathbb{T}[\![\texttt{Piecewise}((t_i\ e_i)_{i=1}^m)]\!] &\coloneqq \lambda r'.\, \textbf{if } \left[(\downarrow^{\text{Real}}_{\text{Outcome}} r') \in \mathbb{V}[\![\mathbb{E}[\![e_1]\!]\,x]\!]\right] \textbf{ then } \mathbb{T}[\![t_1]\!]\,r' \\
&\qquad \textbf{else if } \ldots \\
&\qquad \textbf{else if } \left[(\downarrow^{\text{Real}}_{\text{Outcome}} r') \in \mathbb{V}[\![\mathbb{E}[\![e_m]\!]\,x]\!]\right] \textbf{ then } \mathbb{T}[\![t_m]\!]\,r' \\
&\qquad \textbf{else undefined} \\
&\quad (\text{iff } (vars\ t_1) = \ldots = (vars\ t_m) \\
&\qquad\qquad = (vars\ e_1) = \cdots = (vars\ e_m) \eqqcolon \{x\}
\end{aligned}
$$

Listing 17. Semantics of Transform.

$$
\begin{aligned}
domainof &: \text{Transform} \to \text{Outcomes} \\
domainof\ \texttt{Id}(x) &\coloneqq ((\texttt{\#f}\ {-\infty})\ (\infty\ \texttt{\#f})) \\
domainof\ \texttt{Reciprocal}(t) &\coloneqq ((\texttt{\#f}\ 0)\ (\infty\ \texttt{\#f})) \\
domainof\ \texttt{Abs}(t) &\coloneqq ((\texttt{\#f}\ {-\infty})\ (\infty\ \texttt{\#f})) \\
domainof\ \texttt{Root}(t\ n) &\coloneqq ((\texttt{\#f}\ 0)\ (\infty\ \texttt{\#f})) \\
domainof\ \texttt{Exp}(t\ r_0) &\coloneqq ((\texttt{\#f}\ {-\infty})\ (\infty\ \texttt{\#f})) \\
domainof\ \texttt{Log}(t\ r_0) &\coloneqq ((\texttt{\#f}\ 0)\ (\infty\ \texttt{\#f})) \\
domainof\ \texttt{Poly}(t\ r_0\ \ldots\ r_m) &\coloneqq ((\texttt{\#f}\ {-\infty})\ (\infty\ \texttt{\#f})) \\
domainof\ \texttt{Piecewise}((t_i\ e_i)_{i=1}^m) &\coloneqq union\ [(intersection\ (domainof\ t_i)\ (\mathbb{E}[\![e]\!]\,x)]_{i=1}^m \\
&\quad \text{where } \{x\} \coloneqq vars\, t_1
\end{aligned}
$$

Listing 18. *domainof* returns the Outcomes on which a Transform is defined.

```
$preimg\ t\ v\ \coloneqq preimage'\ t\ (intersection\ (domainof\ t)\ v)$
$preimage'\ \mathtt{Id}\ v \coloneqq v$
$preimage'\ t\ \varnothing \coloneqq \varnothing$
$preimage'\ t\ (v_1 \amalg \cdots \amalg v_m) \coloneqq union\ (preimg\ t\ v_1)\ \ldots\ (preimg\ t\ v_m)$
$preimage'\ t\ \{r_1 \ldots r_m\} \coloneqq preimg\ t'\ (union\ (finv\ t\ r_1)\ \ldots\ (finv\ t\ r_m))$
$preimage'\ t\ ((b_{\text{left}}\ r_{\text{left}})\ (r_{\text{right}}\ b_{\text{right}})) \coloneqq \textbf{match}\ t$
 $\triangleright\ \mathtt{Radical}(t'\ n) \mid \mathtt{Exp}(t'\ r) \mid \mathtt{Log}(t'\ r) \Rightarrow \textbf{let}\ \{r'_{\text{left}}\}\ \textbf{be}\ finv\ t\ r_{\text{left}}$
                                             $\textbf{in let}\ \{r'_{\text{right}}\}\ \textbf{be}\ finv\ t\ r_{\text{right}}$
                                             $\textbf{in}\ preimg\ t'\ ((b_{\text{left}}\ r'_{\text{left}})\ (r'_{\text{right}}\ b_{\text{right}}))$
 $\triangleright\ \mathtt{Abs}(t') \Rightarrow \textbf{let}\ v'_{\text{pos}}\ \textbf{be}\ ((b_{\text{left}}\ r_{\text{left}})\ (r_{\text{right}}\ b_{\text{right}}))$
                $\textbf{in let}\ v'_{\text{neg}}\ \textbf{be}\ ((b_{\text{right}}\ -r_{\text{right}})\ (-r_{\text{left}}\ b_{\text{left}}))$
                $\textbf{in}\ preimg\ t'\ (union\ v'_{\text{pos}}\ v'_{\text{neg}})$
 $\triangleright\ \mathtt{Reciprocal}(t') \Rightarrow \textbf{let}\ \langle r'_{\text{left}}, r'_{\text{right}}\rangle\ \textbf{be if}\ (0 \le r_{\text{left}} < r_{\text{right}})$
                                        $\textbf{then}\ \langle \textbf{if}\ (0 < r_{\text{left}})\ \textbf{then}\ 1/r_{\text{left}}\ \textbf{else}\ \infty,$
                                            $\textbf{if}\ (r_{\text{right}} < \infty)\ \textbf{then}\ 1/r_{\text{right}}\ \textbf{else}\ 0\rangle$
                                        $\textbf{else}\ \langle \textbf{if}\ (-\infty < r_{\text{left}})\ \textbf{then}\ 1/r_{\text{left}}\ \textbf{else}\ 0,$
                                            $\textbf{if}\ (r_{\text{right}} < 0)\ \textbf{then}\ 1/r_{\text{right}}\ \textbf{else}\ -\infty\rangle$
                     $\textbf{in}\ preimg\ t'\ ((b_{\text{right}}\ r'_{\text{right}})\ (r'_{\text{left}}\ b_{\text{left}}))$
 $\triangleright\ \mathtt{Polynomial}(t\ r_0\ \ldots\ r_m) \Rightarrow \textbf{let}\ v'_{\text{left}}\ \textbf{be}\ polyLte\ \neg b_{\text{left}}\ r_{\text{left}}\ r_0\ \ldots\ r_m$
                                  $\textbf{in let}\ v'_{\text{right}}\ \textbf{be}\ polyLte\ b_{\text{right}}\ r_{\text{right}}\ r_0\ \ldots\ r_m$
                                  $\textbf{in}\ preimg\ t'\ (intersection\ v'_{\text{right}}\ (complement\ v'_{\text{left}}))$
 $\triangleright\ \mathtt{Piecewise}((t_i\ e_i)_{i=1}^m) \Rightarrow \textbf{let}_{1\le i\le m}\ v'_i\ \textbf{be}\ preimg\ t_i\ ((b_{\text{left}}\ r_{\text{left}})\ (b_{\text{right}}\ r_{\text{right}}))$
                                $\textbf{in let}_{1\le i\le m}\ v_i\ \textbf{be}\ intersection\ v'_i\ (\mathbb{E}\ [\![e_i]\!]\ x),$
                                $\textbf{in}\ union\ v_1\ \ldots\ v_m$      where $\{x\} \coloneqq vars\ t_1$
```

Listing 19. *preimg* computes the generalized inverse of a many-to-one Transform.

```
                         $finv : \text{Transform} \to \text{Real} \to \text{Outcomes}$
                    $finv\ \mathtt{Id}(x)\ r \coloneqq \{r\}$
            $finv\ \mathtt{Reciprocal}(t)\ r \coloneqq \textbf{if}\ (r = 0)\ \textbf{then}\ \{-\infty\ \infty\}\textbf{else}\ \{1/r\}$
                   $finv\ \mathtt{Abs}(t)\ r \coloneqq \{-r\ r\}$
                $finv\ \mathtt{Root}(t\ n)\ r \coloneqq \textbf{if}\ (0 \le r)\ \textbf{then}\ \{r^n\}\ \textbf{else}\ \varnothing$
                $finv\ \mathtt{Exp}(t\ r_0)\ r \coloneqq \textbf{if}\ (0 \le r)\ \textbf{then}\ \{\log_{r_0}(r)\}\ \textbf{else}\ \varnothing$
                $finv\ \mathtt{Log}(t\ r_0)\ r \coloneqq \{r_0^r\}$
 $finv\ (\mathtt{Polynomial}\ t\ r_0\ \ldots\ r_m)\ r \coloneqq polySolve\ r\ r_0\ r_1\ \ldots\ r_m$
    $finv\ (\mathtt{Piecewise}\ (t_i\ e_i)_{i=1}^m) \coloneqq union\ [(intersection\ (finv\ t_i\ r)\ (\mathbb{E}\ [\![e_i]\!]\ x))]_{i=0}^m,$
                         where $\{x\} \coloneqq vars\ t_1$
```

Listing 20. *finv* computes the generalized inverse of a many-to-one transform at a single Real.

```
$polyLim : \mathsf{Real}^{+} \rightarrow \mathsf{Real}^{2}$
$polyLim\ r_0 \coloneqq \langle r_0, r_0 \rangle$
$polyLim\ r_0\ r_1\ \dots\ r_m \coloneqq$
    $\textbf{let}\ n\ \textbf{be}\ \max\{j \mid r_j > 0\}$
    $\textbf{in if}\ (even\ n)\ \textbf{then}\ (\textbf{if}\ (r_n > 0)\ \textbf{then}\ \langle \infty, \infty \rangle\ \textbf{else}\ \langle -\infty, -\infty \rangle)$
                 $\textbf{else}\ (\textbf{if}\ (r_n > 0)\ \textbf{then}\ \langle -\infty, \infty \rangle\ \textbf{else}\ \langle \infty, -\infty \rangle)$
```

Listing 21. *polyLim* computes the limits of a polynomial limits at the infinities.

```
$polySolve : \mathsf{Real} \rightarrow \mathsf{Real}^{+} \rightarrow \mathsf{Set}$
$polySolve : r\ r_0\ \dots\ r_m \coloneqq \textbf{match}\ r$
    $\triangleright (\infty \mid -\infty)$   $\Rightarrow \textbf{let}\ \langle r_{\text{neg}}, r_{\text{pos}} \rangle$   $\textbf{be}\ polyLim\ r_0\ \dots\ r_m$
                       $\textbf{in let}\ f$             $\textbf{be}\ \lambda r'.\ \textbf{if}\ (r = \infty)\ \textbf{then}\ (r' = \infty)\ \textbf{else}\ (r' = -\infty)$
                       $\textbf{in let}\ v_{\text{neg}}$          $\textbf{be if}\ (f\ r_{\text{neg}})\ \textbf{then}\ \{-\infty\}\ \textbf{else}\ \varnothing$
                       $\textbf{in let}\ v_{\text{pos}}$          $\textbf{be if}\ (f\ r_{\text{pos}})\ \textbf{then}\ \{\infty\}\ \textbf{else}\ \varnothing$
                       $\textbf{in}$                  $union\ v_{\text{pos}}\ v_{\text{neg}}$
    $\triangleright\ \textbf{else}$          $\Rightarrow (roots\ (r_0 - r)\ r_1\ \dots\ r_m)$
```

Listing 22. *polySolve* computes the set of values at which a polynomial is equal to a specific value r.

```
$polyLte : \mathsf{Boolean} \rightarrow \mathsf{Real} \rightarrow \mathsf{Real}^{+} \rightarrow \mathsf{Outcomes}$
$polyLte\ b\ r\ r_0\ \dots\ r_m \coloneqq \textbf{match}\ r$
    $\triangleright -\infty$   $\Rightarrow \textbf{if}\ b\ \textbf{then}\ \varnothing\ \textbf{else}\ (polySolve\ r\ r_0\ \dots\ r_m)$
    $\triangleright\ \infty$    $\Rightarrow \textbf{if}\ \neg b\ \textbf{then}\ \texttt{((\#t}\ -\infty\texttt{)}\ \texttt{(}\infty\ \texttt{\#t))}$
                 $\textbf{else let}\ \langle r_{\text{left}}, r_{\text{right}} \rangle\ \textbf{be}\ polyLim\ r_0\ \dots\ r_m$
                     $\textbf{in let}\ \langle b_{\text{left}}, b_{\text{right}} \rangle\ \textbf{be}\ \langle r_{\text{left}} = \infty, r_{\text{right}} = \infty \rangle$
                     $\textbf{in}\ \texttt{((}b_{\text{left}}\ -\infty\texttt{)}\ \texttt{(}\infty b_{\text{right}}\texttt{))}$
    $\triangleright\ \textbf{else}$   $\Rightarrow \textbf{let}\ [r_{\text{s},i}]_{i=1}^{k}\ \textbf{be}\ roots\ (r_0 - r)\ r_1\ \dots\ r_m$
        $\textbf{in let}\ [\langle r'_{\text{left},i}, r'_{\text{right},i} \rangle]_{i=0}^{k}\ \textbf{be}\ [\langle -\infty, r_{\text{s},0} \rangle, \langle r_{\text{s},1}, r_{\text{s},2} \rangle, \dots, \langle r_{\text{s},k-1}, r_{\text{s},k} \rangle, \langle r_{\text{s},k}, \infty \rangle]$
        $\textbf{in let}\ f_{\text{mid}}\ \textbf{be}\ \lambda r r'.\ \textbf{if}$      $(r = -\infty)$   $\textbf{then}\ r'$
                               $\textbf{elseif}$   $(r' = \infty)$    $\textbf{then}\ r$
                               $\textbf{else}$     $(r + r')/2$
        $t'\ \textbf{be}\ \texttt{Poly(Id(x)}\ (r_0 - r)\ r_1\ \dots\ r_m\texttt{)}$
        $\textbf{in}\ union \begin{bmatrix} \textbf{if}\ \mathbb{T}[\![t']\!]\ (f_{\text{mid}}\ r'_{\text{left},i}\ r'_{\text{right},i}) & \textbf{then}\ \texttt{((}b\ r'_{\text{left},i}\texttt{)}\ \texttt{(}r'_{\text{right},i}\ b\texttt{))} \\ & \textbf{else}\ \varnothing \end{bmatrix}_{i=0}^{k}$
```

Listing 23. *polyLte* computes the set of values at which a polynomial is less than a given value r.