Application of Mathematical Components to probabilities and program formalization

Reynald Affeldt

Nat. Inst. of Advanced Industrial Science and Technology (AIST), Tokyo, Japan

December 7, 2022
This Talk

- We have been interested in applying formal verification to information security
- We have been led to look at topics such as information theory, error-correcting codes, probabilistic programs, etc.
- We would like to report on a few formal theories that might be of general interest
- The common topic is probability, the work spans several years
- Our message is that MathComp has been providing us with a reliable environment for our experiments
Outline

Information Theory using MathComp

Monadic Equational Reasoning for Probabilistic Programs

Formal Semantics for Statistical Modeling
SSReflect Early Adopters?

- Starting in 2004, we were working on formal proof of imperative programs in Coq using separation logic

- We were facing productivity issues that we failed to analyze correctly

- By chance, we ran into

and realized that many of our concerns were addressed by SSReflect
SSReflect/MathComp in Japan

1. We have been trying to promote MathComp in Japan with lectures
2. These lectures turned into a book in 2018

3. In this book, tactics are given mascots, e.g.:
The **SSReflect Zoo According to** **move** **view** **case** **rewrite** **elim** **apply**
Outline

Information Theory using MathComp

Monadic Equational Reasoning for Probabilistic Programs

Formal Semantics for Statistical Modeling
Fundamental Questions Answered Information Theory

1. What is the ultimate data compression rate\(^1\)?
2. What is the ultimate encoding rate\(^2\) for communication?

Setting:
- a source is a probability distribution
- a channel is a stochastic matrix (e.g., the binary symmetric channel \(\begin{bmatrix} 1-p & p \\ p & 1-p \end{bmatrix}\))

Shannon answered the fundamental questions in 1948:
1. **Source coding theorem:** One cannot minimize the compression rate below the *entropy*
2. **Channel coding theorem:** One cannot maximize the encoding rate beyond the *capacity* of a channel

\(^1\)compressed bitstring size / original message size  
\(^2\)original message size / encoded message size
Information Theory: Basic Definitions [CT01]

- The **entropy** of a random variable taking $n$ values with probabilities $p_i$ is $H = - \sum_{i=0}^{n-1} p_i \log p_i$

- Given two probability mass functions $p$ and $q$, the **divergence** is $D(p||q) = \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)}$

- The **mutual information** between two random variables $X$, $Y$ with joint probability mass function $p(x, y)$ and marginals $p(x)$ and $p(y)$ is $I(X; Y) = D(p(x, y)||p(x)p(y))$

- Consider an input $X$ and an output $Y$. A channel is a conditional probability distribution $p_{Y|X}(y|x)$. The **capacity** is $C = \max_{p(x)} I(X; Y)$, $p(x)$ ranging over all the input distributions

$\Rightarrow$ It looks like a good fit for SSReflect’s iterated operators [BGBP08]
Finite Probabilities with \texttt{SSReflect} and \texttt{Coq}

(We are in 2009)

Finite probability theory with the iterated operators and the finite sets of \texttt{MathComp} and the real numbers of \texttt{Coq}:

- Distributions over a finite type:

  \begin{verbatim}
  Record fdist (A : finType) := mk { 
      f := A -> R+ ; 
      _ := \sum_(a in A) f a == 1 :> R }.
  \end{verbatim}

- Probability of an event $E$ given a distribution $P$:

  \begin{verbatim}
  Definition Pr P (E : \{set A\}) := \sum_(a in E) P a.
  \end{verbatim}

- Random variables:

  \begin{verbatim}
  Definition RV U T (P : fdist U) := U -> T.
  \end{verbatim}

  - Distribution of a random variable $X$: $\`Pr[ X = a ]$, shortcut for $Pr P (X @^- {1: [set a]}$ or for $fdistmap X P a$
Information Theory with MathComp

Overview

Starting from:

- \( H = - \sum_{i=0}^{n-1} p_i \log p_i \)
  
  **Definition** entropy := \(- \sum_{a \in A} P_a * \log (P_a)\).

- \( \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)} \)
  
  **Definition** div := \( \sum_{a \in A} P_a * \log (P_a / Q_a) \).

- \( D(p(x,y) \| p(x)p(y)) \)
  
  **Definition** mutual_info := \( D(PQ \| \| P^1 \ x \ PQ^2) \).

We completely formalized an introductory textbook to information theory:

- Shannon’s theorems [AH12, AHS14]
- Error-correcting codes (Hamming, BCH, Reed-Solomon, LDPC) [AG15, AGS20b]
- Presentations to information theorists [OHA14, AGS16, AGS18]
Convexity of Information-theoretic Definitions

Statements from [CT01]

Theorem

$H(p)$ is a concave function of $p$.

Theorem

$D(p||q)$ is convex in the pair $(p, q)$, i.e., if $(p_1, q_1)$ and $(p_2, q_2)$ are pairs of probability mass functions, then

$$D(\lambda p_1 + (1 - \lambda) p_2 || \lambda q_1 + (1 - \lambda) q_2) \leq \lambda D(p_1 || q_1) + (1 - \lambda) D(p_2 || q_2)$$

for all $0 \leq \lambda \leq 1$.

Theorem

Let $(X, Y) \sim p(x, y) = p(x)p(y|x)$. The mutual information $I(X; Y)$ is a concave function of $p(x)$ for fixed $p(y|x)$ and a convex function of $p(y|x)$ for fixed $p(x)$.
Convex Space

A *convex space* is a carrier together with a family of binary operators $a \triangleleft p \triangleright b$ with $0 \leq p \leq 1$ such that [Sto49, Fri09]:

- $a \triangleleft 0 \triangleright b = b$
- $a \triangleleft p \triangleright a = a$ (idempotence)
- $a \triangleleft p \triangleright b = b \triangleleft 1 - p \triangleright a$ (skewed commutativity)
- $a \triangleleft p \triangleright (b \triangleleft q \triangleright b) = (a \triangleleft r \triangleright b) \triangleleft s \triangleright c$ (quasi-associativity)

with $s = \overline{pq}$ and $r = \frac{p}{s}$ (where $\overline{x} = 1 - x$)

Examples: real numbers $(pa + (1 - p)b)$, functions to a convex space, finite distributions, etc.

Allows for generic definitions ($U$ convex space, $V$ ordered convex space):

- $f : U \to U'$ is affine $\overset{\text{def}}{=} \forall a, b, 0 \leq p \leq 1, f(a \triangleleft p \triangleright b) = f(a) \triangleleft p \triangleright f(b)$
- $f : U \to V$ is convex $\overset{\text{def}}{=} \forall a, b, 0 \leq p \leq 1, f(a \triangleleft p \triangleright b) \leq f(a) \triangleleft p \triangleright f(b)$
- also convex sets and hulls
Convex Space in MathComp

Conveniently defined using Hierarchy-Builder [CST20]:

1. Declare an interface:

   ```
   HB.mixin Record isConvexSpace (T : Type) := {
     _ <| _ |> _ : forall p, T -> T -> T ;
     conv1 : forall a b, a <| 1%:pr |> b = a ;
     convmm : forall p a, a <| p |> a = a ;
     convC : forall p a b, a <| p |> b = b <| p.%:pr |> a ;
     convA : forall (p q : prob) (a b c : T),
     a <| p |> (b <| q |> c) =
     (a <| [r_of p, q] |> b) <| [s_of p, q] |> c }.
   ```

2. Declare a structure:

   ```
   #[short(type=convType)]
   HB.structure Definition ConvexSpace := {T of isConvexSpace T }.
   ```

3. Build instances: any lmodType (and thus real numbers), the type “fdist A”, the type “A -> fdist B”, etc.
Short statements for convexity properties of information theoretic definitions:

- **Lemma** entropy_concave :
  \[
  \text{concave}\_\text{function}\ (\text{fun } P : \text{fdist } A \Rightarrow \mathcal{H} P).
  \]

- **Lemma** mutual_information_concave W :
  \[
  \text{concave}\_\text{function}\ (\text{fun } P \Rightarrow \text{mutual}\_\text{info} (P \times X W)).
  \]

  where \(P \times X W\) is the product distribution \(\lambda(x,y).P x \cdot W x y\)

- **Lemma** mutual_information_convex P :
  \[
  \text{convex}\_\text{function}\ (\text{fun } W : A \Rightarrow \text{fdist } B \Rightarrow \text{mutual}\_\text{info} (P \times X W)).
  \]
Real Cones
A practical tool to reason about convexity

- In a convex space, quasi-associativity and skewed commutativity make for cumbersome symbolic computations
- It is actually possible to transpose such computations into real cones where addition is commutative and associative [VW06]:

```plaintext
HB.mixin Record isQuasiRealCone A := {
    addpt : A -> A -> A ;
    zero : A ;
    addptC : commutative addpt ;
    addptA : associative addpt ;
    addpt0 : right_id zero addpt ;
    scalept : R -> A -> A ;
    scale0pt : forall x, scalept 0 x = zero ;
    scale1pt : forall x, scalept 1 x = x ;
    scaleptDr : forall r, {morph scalept r : x y / addpt x y >-> addpt x y} ;
    scaleptA : forall p q x, 0 <= p -> 0 <= q ->
        scalept p (scalept q x) = scalept (p * q) x }.

HB.mixin Record isRealCone A of isQuasiRealCone A := {
    scaleptDl : forall p q x, 0 <= p -> 0 <= q ->
        scalept (p + q) x = addpt (scalept p x) (scalept q x) }.
```
Consider the following inductive type:

\[
\text{Inductive scaled (A : Type) := Scale of Rpos & A | Zero.}
\]

When A is a convex space:

- scaled A can be equipped with a real cone structure (take \(\text{addpt\ (Scaled\ r\ x)\ (Scaled\ q\ y)\ to\ be\ \(r + q)\left(x \triangleleft \frac{r}{r+q} \triangleright y\right)\)\)

- scaled A can be equipped with a convex space structure (take \(x \triangleleft p \triangleright y\) to be \(\text{addpt\ (scalept\ p\ x)\ (scalept\ (1 - p)\ y)}\) [VW06])

We can transpose symbolic computations using the fact that Scale 1 is injective and affine:

- \(a \triangleleft p \triangleright b\)
  - \(\rightarrow \text{Scaled 1} a \triangleleft p \triangleright \text{Scaled 1} b\)
  - \(\rightarrow \text{addpt\ (scalept\ p\ (Scaled 1 a))\ (scalept\ (1 - p)\ (Scaled 1 b))}\)
    - where addition is associative and commutative

See InfoTheo online or [AGS20a] for details.
Outline

Information Theory using MathComp

Monadic Equational Reasoning for Probabilistic Programs

Formal Semantics for Statistical Modeling
Monadic Equational Reasoning

This is an approach to verify programs with effects using equational reasoning [GH11]

▸ effects are represented by monad interfaces with typically:
  ▸ an operator (failure, arbitrary choice, probabilistic choice, etc.)
  ▸ rewriting laws in the form of equations

▸ monad interfaces can **inherit** from other interfaces and can be **combined**

Our starting idea:

▸ build a hierarchy of interfaces using packed classes [GGMR09]
▸ use **SSReflect**’s **rewrite** [GT12] to perform equational reasoning
Example of Monadic Laws

Reminder

Monad laws (two operators: \( ret(\cdot) \) and \( \cdot \gg\gg \cdot \)):

1. \( \text{ret}(a) \gg\gg f = f \cdot a \) (left neutral)
2. \( m \gg\gg (\lambda x. \text{ret}(x)) = m \) (right neutral)
3. \( (m \gg\gg f) \gg\gg g = m \gg\gg (\lambda x.f \cdot x \gg\gg g) \) (associativity)

Arbitrary choice (one operator: \( \cdot \Box \cdot \)):

1. \( (m_1 \Box m_2) \Box m_3 = m_1 \Box (m_2 \Box m_3) \) (associativity)
2. \( (m_1 \Box m_2) \gg\gg k = (m_1 \gg\gg k) \Box (m_2 \gg\gg k) \)
   (left-distributivity of bind w.r.t. arbitrary choice)

etc.
Functors and Monads with **Hierarchy-Builder**

- We consider Coq’s `Type` to be the category **Set** of sets and functions [TJ16]
- Let us start with functors:
  - action on objects: `F : Type -> Type` (carrier)
  - action on morphisms: `actm` below

```coq
HB.mixin Record isFunctor (F : Type -> Type) := { actm : forall A B, (A -> B) -> F A -> F B; functor_id : FunctorLaws.id actm ; functor_o : FunctorLaws.comp actm }.
```

- Next, monads (ret/bind interface):

```coq
HB.factory Record isMonad_ret_bind (F : Type -> Type) := { ret' : forall A, A -> F A ; bind : forall A B, F A -> (A -> F B) -> F B ; bindretf : BindLaws.left_neutral bind ret' ; bindmret : BindLaws.right_neutral bind ret' ; bindA : BindLaws.associative bind }.```
The Interface of the Probability Monad

Probability monad:

- extends the type of Monad
- similar interface to convex spaces
- with left-distributivity of bind w.r.t. probabilistic choice

```
HB.mixin Record isMonadProb (M : Type -> Type) of Monad M := {
  _ <| _ |>_ _ : forall p T, M T -> M T -> M T ;
  choice0 : forall T a b, a <| 0 |> b = b ;
  choiceC : forall T p a b, a <| p |> b = b <| 1 - p |> a ;
  choicemm : forall T p, idempotent (_ <| p |> _ ) ;
  choiceA : forall T p q r s a b c ,
            p = r * s -> 1 - s = (1 - p) * (1 - q) ->
            a <| p |> (b <| q |> c) = (a <| r |> b) <| s |> c ;

  choice_bindDl : forall p a b ,
                 (a <| p |> b ) >>= f = (a >>= f) <| p |> (b >>= f ) }.
```
Model of the Probability Monad

The interface do have an implementation

- Finite distributions do not form a monad because
  \( \text{fdist} : \text{finType} \rightarrow \text{Type} \) is not an endofunction

- Hence *finitely-supported distributions* with \( \text{finmap} \) [CS15]:

  \[
  \text{Record} \ \text{fsdist} (A : \text{choiceType}) := \text{mk} \ {\}
  \]
  \[
  \ f :> \{\text{fsfun} A \rightarrow R \text{ with } 0\} ; \\
  \ _ : \text{all (fun} \ x \Rightarrow 0 < b f x) (\text{finsupp} f) && \ \\
  \ \sum_{(a \leftarrow \text{finsupp} f) f a == 1} {}. \]

- The required operators (\( \text{ret}() \), \( \cdot \gg \cdot \), \( \cdot \land \cdot \land \cdot \land \cdot \)):
  - \( \text{fsdist1} : \forall A : \text{choiceType}, A \rightarrow \{\text{dist} A\} \)
    \[
    \text{def} = \{\text{fsfun} b \text{ in } [\text{fset} a] \Rightarrow 1 \mid 0\} \]
  - \( \text{fsdistbind} : \forall A B : \text{choiceType}, \{\text{dist} A\} \rightarrow (A \rightarrow \{\text{dist} B\}) \rightarrow \{\text{dist} B\} \)
    \[
    \text{def} = \lambda b. \sum_{a \in \text{supp}(d)} d(a) \times (f(a))(b) \text{ over } \bigcup_{x \in f(\text{supp}(d))} \text{supp}(x) \]
  - \( \text{fsdistConv} : \forall A : \text{choiceType}, \text{prob} \rightarrow \{\text{dist} A\} \rightarrow \{\text{dist} A\} \rightarrow \{\text{dist} A\} \)
    \[
    \text{def} = \lambda a. p \ d_1(a) + (1 - p) \ d_2(a) \text{ over } \text{supp}(d_1) \cup \text{supp}(d_2) \]
The Start of a Hierarchy of Effects

- quasi-real cones
- real cones
- convex spaces
- probMonad

Solid arrow: inherits
Dotted arrow: uses
Probabilistic Program Verification using Rewriting

A biased coin with probability $p$:

```
Definition bcoin {M : probMonad} p : M bool := Ret T <| p |> Ret F.
```

Simple statement:

```
Definition two_coins p q : M (bool * bool) :=
  do a <- bcoin p;
  do b <- bcoin q;
  Ret (a, b).
```

Lemma `two_coinsE` $p q : two_coins p q = two_coins q p$.

Proof:

```
rewrite /two_coins /bcoin.
(Ret T <|p|> Ret F) >>=
  (fun a => (Ret T <|q|> Ret F) >>= (fun b => Ret (a, b)))
rewrite ![in LHS](choice_bindDl,bindretf).
(* choice_bindDl -> probability monad law *)
(* bindretf = ret x >>= f = f x -> monad law *)
(Ret (T, T) <|q|> Ret (T, F)) <|p|> (Ret (F, T) <|q|> Ret (F, F))
rewrite -choiceACA.
(* interchange <|p|> <|q|> -> real cones *)
(Ret (T, T) <|p|> Ret (T, F)) <|q|> (Ret (F, T) <|p|> Ret (F, F))
...
Examples Formalized with The **MONAE** Library

- tree relabeling [GH11], Spark aggregation [Mu19b], Monty-Hall problem [GH11, Gib12]
- n-queens [GH11], completed by [Mu19a] (we fixed an earlier version of the latter)
- quicksort [MC20] (we completed a pre-existing formalization in Agda)
- Jaskelioff’s theory of modular monad transformers [Jas09] (we actually proposed a fix for this theory)

Experiments documented in the following papers [ANS19, AN21, AGNS21, SA22]
Combination of Monad Interfaces Can be Difficult

It was observed in [ASCG16] that [GH11] contains a mistake\(^3\):

- right-distributivity of bind over probabilistic choice
  \[m \gg = \lambda x.(k \times \triangleleft p \triangleright k' \times) = (m \gg = k) \triangleleft p \triangleright (m \gg = k')\]

- combined with

- distributivity of probabilistic choice over arbitrary choice
  \[m \triangleleft p \triangleright (a \Box b) = (m \triangleleft p \triangleright a) \Box (m \triangleleft p \triangleright b)\]

result in a degenerated theory:

- distributivity of arbitrary choice over probabilistic choice
  \[m \Box (a \triangleleft p \triangleright b) = (m \Box a) \triangleleft p \triangleright (m \Box b)\]

- which implies
  \[a \triangleleft p \triangleright b = a \triangleleft q \triangleright b\]

\(^3\)We checked with Monae that [GH11] was not relying on this mistake.

⇒ It is important to provide implementations for interfaces.
The probDrMonad adds:

- \( m \gg \lambda x. (k x \triangleleft p \triangleright k' x) = (m \gg k) \triangleleft p \triangleright (m \gg k') \)

The geometrically convex monad adds:

- \( m \triangleleft p \triangleright (a \square b) = (m \triangleleft p \triangleright a) \square (m \triangleleft p \triangleright b) \)
Model of the Geometrically Convex Monad

What is a computation in this monad?

▷ Gibbons observes that it should be a convex-closed sets of probability distributions [Gib12]
▷ Cheung provides a construction using adjunctions between categories [Che17]

We formalized Cheung’s construction [AGNS21]:

This relies on an extension of Monae with concrete categories (to go beyond Set)

Ask Takafumi here in this room!
Outline

Information Theory using MathComp

Monadic Equational Reasoning for Probabilistic Programs

Formal Semantics for Statistical Modeling
Example: guessing whether or not today’s a weekday by looking at the number of buses passing by [Sta20]

\[
\text{normalize (}
\quad \text{let } x = \text{sample (bernoulli (2 / 7)) in}
\quad \text{let } r = \text{if } x \text{ then } 3 \text{ else } 10 \text{ in}
\quad \text{let } _ = \text{score (r} ^ 4 / 4! * e ^ (- r)) \text{ in}
\quad \text{return } x)
\]

Intuitive explanation:

- \text{sample} takes a probability measure
- \text{normalize} returns a probability measure
- \text{score (f x)} means that we observe x from the distribution corresponding to the density f
  - here, observe 4 from the Poisson distribution (of density \( \frac{r^k}{k!} e^{-r} \))

Problem: existing formalizations in \text{CoQ} use axioms [HcS19, ZA22]
Formalization of Kernels using MathComp-Analysis

Staton proposed a semantics for programs with sampling, scoring, and normalization using \textit{s-finite kernels} [Sta17]

Definition:

\begin{itemize}
  \item A \textit{kernel} $X \rightsquigarrow Y$ is a function $k : X \to \Sigma_Y \to [0, \infty]$ such that
    \begin{enumerate}
      \item for all $x$, $k \times x$ is a measure
      \item for all measurable set $U$, $x \mapsto k \times U$ is measurable
    \end{enumerate}
\end{itemize}

Reminder: measure theory in MathComp-Analysis [AC22]

\begin{tabular}{|c|c|}
  \hline
  measurable spaces & type \texttt{measurableType} \\
  \hline
  measure & type \{\texttt{measure \ set T \to \bar{\mathbb{R}}}\} \\
  \hline
  measurable functions & predicate \texttt{measurable_fun} \\
  \hline
\end{tabular}

Formal definition of kernel (notation $R.-ker X \rightsquigarrow Y$):

\begin{verbatim}
HB.mixin Record isKernel
  X Y R (k : X -> \{\texttt{measure \ set Y \to \bar{\mathbb{R}}}\}) :=
  \{ measurable_kernel :
    forall U, measurable U -> measurable_fun setT (fun x => k x U) \}.
\end{verbatim}
S-Finite and Finite Kernels

A circular-looking definition

Definition:

- A kernel $k : X \sim Y$ is finite when $\exists r$ s.t. $\forall x$, $k \times Y < r$ (uniformly upper bounded)
- A kernel $k$ is s-finite when there exists a sequence of finite kernels $s$ such that $k = \sum_{i=0}^{\infty} s_i$

Circularity?

- s-finite kernels are more general than finite kernels (so they should be defined first)
- finite kernels are needed to define s-finite kernels...
Wanted: Hierarchy of Kernels

To implement Staton’s semantics of probabilistic programs

Kernel $X \sim Y$

S-finite kernel $X \rightsquigarrow_{s\text{-}fin} Y$

Finite kernel $X \rightsquigarrow_{\text{fin}} Y$

Subprobability kernel $X \rightsquigarrow_{\text{subprob}} Y$

Probability kernel $X \rightsquigarrow_{\text{prob}} Y$
S-Finite and Finite Kernels

A recipe using Hierarchy-Builder

1. Interface for s-finite kernels using a predicate for finite kernels:

   HB.mixin Record Kernel_isSFinite_subdef
   X Y R (k : X -> {measure set Y -> \bar R}) := {
   sfinite_subdef : exists2 s : (R.-ker X ~> Y)^nat,
   forall n, measure_fam_uub (s n) &
   forall x U, measurable U -> k x U = kseries s x U }.

   Notation: R.-sfker X ~> Y, inherits from R.-ker X ~> Y

2. Interface for finite kernels:

   HB.mixin Record SFiniteKernel_isFinite
   X Y R (k : X -> {measure set Y -> \bar R}) :=
   { measure_uub : measure_fam_uub k }.

   Notation: R.-fker X ~> Y, inherits from R.-sfker X ~> X

3. Definitive interface for s-finite kernels:

   HB.factory Record Kernel_isSFinite
   X Y R (k : X -> {measure set Y -> \bar R})
   of isKernel _ _ _ _ _ k := {
   sfinite : exists s : (R.-fker X ~> Y)^nat,
   forall x U, measurable U -> k x U = kseries s x U }.
The main property of s-finite kernels is that they are stable by composition (this provides a semantics for \texttt{let x := e in e'}).

- Given \( l : X \sim Y \) and \( k : X \times Y \sim Z \), the composition \( l ; k \) is defined by

\[
\lambda x \in X. \int_y k(x, y) \, U(dl x)
\]

- Reminder: integral theory in \texttt{MathComp-Analysis} [AC22]

\[
\int_{x \in A} f(x)(d \mu) \quad \int[mu]_(x \in A) f x
\]

- Formal definition of composition:

\texttt{Definition kcomp l k x U := \int[l x]_y k(x, y) U.}

- Staton proved that the composition of s-finite kernels is a s-finite kernel [Sta17]. He skipped the proof that it is a kernel. It is not trivial but it can be achieved it by adapting existing lemmas from Fubini’s theorem available in \texttt{MathComp-Analysis}.
Semantics of Sampling using S-finite Kernels

For illustration

What is the semantics of \texttt{sample (bernoulli (2 / 7))}? 

1. Build the measurable space of probability measures \( p\text{probability} \ Y \ \mathbb{R} \)
   - generated from the set of probability measures \( \mu \) such that \( \mu(U) < r \) for all measurable sets \( U \) and \( 0 \leq r \leq 1 \)
   - The type \( X \rightarrow p\text{probability} \ Y \ \mathbb{R} \) is essentially \( X \rightarrow \{\text{measure set} \ Y \rightarrow \overline{\mathbb{R}}\} \)

2. \( P : X \rightarrow p\text{probability} \ Y \ \mathbb{R} \) is a kernel
   - for any measurable set \( U \), \texttt{fun} \( x \rightarrow P \ x \ U \) is measurable

3. \( P : X \rightarrow p\text{probability} \ Y \ \mathbb{R} \) is a probability kernel
   - because for all \( x \), \( P \ x \ \text{set}\text{T} = 1 \)
   - it is therefore automatically a s-finite kernel

4. For our example, take for \( P \) the (constant) Bernoulli probability measure (built out of Dirac measures)
normalize (let x = sample (bernoulli (2 / 7)) in
let r = if x then 3 else 10 in
let _ = score (r ^ 4 / 4! * e ^ (- r)) in
return x)

⇓

Definition kstaton_bus : R.-sfker T ~> mbool :=
letin (sample (bernoulli p27))
(letin
 (letin (ite var2of2 (ret k3) (ret k10))
   (score (measurable_fun_comp mh var3of3)))
 (ret var2of3)).
(* NB: density function parameterized,
"De Bruijn indices" for variables *)
Definition staton_bus := normalize kstaton_bus.
Symbolic Evaluation of Statistical Models

We can evaluate a model to a distribution:

Lemma staton_busE P (t : R) U :
  let N := ((2 / 7) * poisson4 3 + (5 / 7) * poisson4 10)%R in
  staton_bus mpoisson4 P t U =
  ((2 / 7)%:E * (poisson4 3)%:E * \d_true U +
   (5 / 7)%:E * (poisson4 10)%:E * \d_false U) * N^-1%:E.

(Proof by rewriting)
In mathematical notation:

\[
\frac{2 \cdot 3^4}{7 \cdot 4!} e^{-3} \delta_1 + \frac{5 \cdot 10^4}{7 \cdot 4!} e^{-10} \delta_0 = 0.780369 \delta_1 + 0.219631 \delta_0
\]

So it is more likely that we are in the weekend
Commutativity Property of Probabilistic Programs

The main motivation for Staton's work

Is the following program transformation correct?

\[
\begin{align*}
&\text{let } x := t \text{ in} \quad \leftrightarrow \quad \text{let } y := u \text{ in} \\
&\text{let } y := u \text{ in} \quad \leftrightarrow \quad \text{let } x := t \text{ in} \\
&\text{ret } (x, y) \quad \leftrightarrow \quad \text{ret } (x, y)
\end{align*}
\]

This is a consequence of Tonelli-Fubini's theorem for \textit{s-finite measures}:

\((* \ f \ \text{measurable non-negative, } m_1, m_2 \ \text{s-finite } *)\)

Lemma sfinite_fubini :

\[
\int_{m_1} x \ \int_{m_2} y \ f (x, y) = \\
\int_{m_2} y \ \int_{m_1} x \ f (x, y).
\]

(This is a consequence of Tonelli-Fubini's theorem for \textit{\(\sigma\)-finite measures}—the one you find in a standard undergraduate textbook on integration).
Conclusion

The MathComp project has been providing us with

- good tactic support
  (e.g., rewrite in monadic equational reasoning)
- a rich, stable, flexible framework (we could combine MathComp and the Coq standard library)
- libraries (finmap, MathComp-Analysis)
- methodologies (packed classes, naming conventions)
- tools (Hierarchy-Builder)

which let us

- develop original formalizations (InfoTheo, Monae)
- develop libraries for (probabilistic) program verification
- fix existing pencil-and-paper proofs
- retrofit results to MathComp (in particular MathComp-Analysis)
[AC22] Reynald Affeldt and Cyril Cohen, Measure construction by extension in dependent type theory with application to integration, Sep 2022.


