Reasoning with Conditional Probabilities and Joint Distributions in Coq

Reynald Affeldt Jacques Garrigue Takafumi Saikawa

Probabilities occur in many applications of computer science, such as communication theory and artificial intelligence. These are critical applications that require some form of verification to guarantee the quality of their implementations. Unfortunately, probabilities are also the typical example of a mathematical theory whose abuses of notations make pencil-and-paper proofs difficult to formalize. In this paper, we experiment a new formalization of conditional probabilities that we validate with two applications. First, we formalize the foundational definitions and theorems of information theory, extending previous work with new lemmas. Second, we formalize the notion of conditional independence and its properties, paving the road for a formalization of graphical probabilistic models.

1 Introduction

Probabilities occur in many applications of computer science. One finds them in information theory to reason about the size of compressed data, in quantitative information flow analysis, in the analysis of side-channel attacks, etc. They are also omnipresent in artificial intelligence. These are critical applications that require some form of verification to guarantee the quality of their implementations.

Formal verification using proof-assistants has emerged as a trustful technique to guarantee the correctness of important mathematical theorems (e.g., [12]), as well as the correctness of critical computer software (e.g., [17]). Regarding more specifically mathematics, formal verification (hereafter, we also say *formalization*) consists in turning a proof from a textbook or a scientific paper (hereafter, we refer to such proofs as "pencil-and-paper proofs") into the language of the proof-assistant. The latter is a software mechanization of some mathematical foundations, such as set theory, higher-order logic, or type theory (as it is the case for the Coq proof-assistant). The success of this transcription guarantees that the original proof was indeed correct but it requires a lot of *proof-engineering*: make explicit all assumptions (in particular hidden ones), build libraries of intermediate definitions and lemmas, find abstractions to reduce complexity, etc.

Pencil-and-paper proofs about probabilities are in particular not easily amenable to formal verification. The **first reason** is that probabilities deal with a wide range of mathematics (from combinatorics to real analysis), but, as of today, no proof-assistant has enough libraries to bring the practitioner the same flexibility as pencil-and-paper

This is a preprint of an article to appear in 2020 in *Computer Software*, the journal of the Japan Society for Software Science and Technology

Coq による条件付き確率と同時分布の形式化とその応用

Reynald Affeldt, 産業技術総合研究所, National Institute of Advanced Industrial Science and Technology.

Jacques Garrigue, 名古屋大学, Graduate School of Mathematics, Nagoya University.

Takafumi Saikawa, 名古屋大学, Graduate School of Mathematics, Nagoya University/Peano System Inc..

proofs. The "ideal" approach (in the sense of the most faithful to the original pencil-and-paper proof) seems to be to derive the formalization of probabilities from a formalization of measure theory and Lebesgue integration. Unfortunately, not all proof assistants have such formalizations available (at the time of this writing, this is work in progress for Coq). There is one in HOL [19] and one in Isabelle/HOL [14], but these proof-assistants are lacking the theories for combinatorics and manipulation of iterated operators provided by the MathComp library of the Coq proof-assistant. Actually, one could even say that what is a good formalization of real analysis is still a topic of research (see for example [1]). The "ideal" approach thus seems out of reach for now but, on the other hand, many applications of information theory target the manipulation of data in computers (like in artificial intelligence) which are always finite. Such applications can be verified by formalizing the finite part of probability and information theory, so that, in our case, a full-fledged formalization of measure theory and Lebesgue integration would look like a sledgehammer to crack a nut. There therefore seems to be much value in exploring a mechanization of discrete probabilities that allows for formal reasoning without this detour. Though arguably a substantial simplification, this does not provide an immediate solution. The **second reason** why formalization of probabilities is difficult is that probabilities are the typical example of a mathematical theory where abuses of notations are omnipresent. Indeed, it is not obvious how to recover in the formalization the terseness of the presentation of information theory or artificial intelligence textbooks. The latter typically feature many implicit assumptions; the main example is the omission of the underlying distributions when talking about probabilities, often ruled out as "obvious from context". In the formalization, we have to fill in all such implicit assumptions. Though this could be done explicitly by the user, we could also program the proof-assistant to automatically do that. Our formalization includes instances of such automation.

In this paper, we focus on the problem of formalizing and using conditional probabilities and joint distributions. Concretely, we extend an existing formalization of information theory [7] with an "explicit" formalization of conditional probability. We say "explicit" because the previous work [7] we extend was already dealing with conditional probabilities but given in the form of stochastic matrices, as used to model channels in information theory (see Sect. 5 for a more precise explanation). To show that our extension is indeed useful, we validate it with two substantial applications:

- First, we formalize the foundations of information theory. This is not a new topic but we are able to improve on previous work by formalizing lemmas there were not formalized before. In fact, we are on the verge of completing the formalization of [Chapter 2 of [9]], which provides the basic definitions of *the* standard textbook for information theory.
- Second, we formalize the notion of conditional independence. We validate this definition by proving the so-called *graphoid axioms* [21]. We are not aware of a previous attempt to formalize the graphoid axioms from the ground up, but they are key to reason about probabilistic graphical models used in artificial intelligence.

Paper Outline

In Sect. 2, we explain how we formalize distributions, joint distributions, and conditional probabilities. In Sect. 3, we explain our formalization of information theory. In Sect. 4, we explain our formalization of conditional independence. We review related work in Sect. 5 and conclude in Sect. 6.

About Notations

This paper displays Coq code verbatim. We rely on the MathComp library whose notations are explained when used for the first time. We enforced a uniform naming convention for variables. We use `A`, `B`, `C`, `D`, etc. for finite types. We use `a`, `b`, etc. for elements of resp. `A`, `B`, etc. We use `E : {set A}`, `F : {set B}`, `G : {set C}`, `H : {set D}`, etc. for events. We use `P`, `Q`, etc. to range over distributions. We use `X`, `Y`, `Z`, `W`, etc. for random variables with values in `A`, `B`, `C`, `D`, resp.

2 Formalization of Conditional Probability

2.1 Background: Formalization of Finite Distributions

We define a finite distribution as a function `f` from a finite set `A` to the set $\mathbb{R}^+$ of nonnegative real numbers, such that the sum of the values of `f` equals 1. The formal definition in Coq is as follows (it comes from previous work [6]):

```
(* Module FDist *)
Record t := mk {
  f :> A -> R+ ;
  _ : \sum_(a in A) f a == 1}.
```

The type `A` has type `finType`, it is a type with a finite number of elements [11]. The type `A -> R+` is for functions with domain `A` and real, nonnegative outputs. The notation `\sum_(a in A)` (where `a` is a binder) is for the sum over `A`. Hereafter, the notation `{fdist A}` causes the type inference to correctly identify `A` as a `finType` *even when* `A` is a composite `finType` (e.g., the product of two finite types `A1 * A2`).

Given a distribution `P : {fdist A}`, we can formalize an event as a finite set `E` over `A` (i.e., a Boolean predicate over `A` represented as a list, whose type is denoted `{set A}` [18]) and compute its probability by summing the individual probabilities of its elements:

```
Variables (A : finType) (P : {fdist A}).
Definition Pr (E : {set A}) :=
  \sum_(a in E) P a.
```

2.2 Joint Distributions

Using the definition of Sect. 2.1, a joint distribution can be represented by a distribution over a product type such as `{fdist A * B}` or `{fdist 'rV[A]_n}` (where `'rV[A]_n` represents the type of row vectors of size `n` [18]).

Given a joint distribution, we often need to consider marginal distributions. For that purpose, it is useful to consider *probability monad*.

2.2.1 The Map Distribution from the Probability Monad

We compute most marginal distributions using the *map distribution*. Given a distribution P of type `{fdist A}` and a function g of type `A -> B`, the map distribution is the distribution of type `{fdist B}` with probability mass function $b \mapsto \sum_{\substack{a \in A \\ g\,a = b}} P\,a$. In practice, this distribution is defined using the probability monad with unit `FDist1.d` and bind operator `FDistBind.d` [4]:

```
(* Module FDistMap. *)
Variables (A B : finType)
          (p : {fdist A}) (g : A -> B).
Definition d : {fdist B} :=
  FDistBind.d p (fun a => FDist1.d (g a)).
```

As hinted at by the comment, this definition occurs inside the module `FDistMap`, which we use as a namespace; hence, the distribution `d` hereafter appears as `FDistMap.d`.

2.2.2 Joint Distributions over a Product Type

Given a joint distribution over a product type, `Bivar.fst` (`Bivar` for “bivariate”) builds the left marginal. It is defined as follows:

```
(* Module Bivar *)
Variables (A B : finType) (P : {fdist A * B}).
Definition fst : {fdist A} := FDistMap.d fst P.
```

Regarding joint distributions over a product type, one might also want to build the right marginal `Bivar.snd` (which is of course defined similarly) and the distribution `Swap.d` that swaps the left and the right elements.

Table 1 summarizes the distributions that one might want to build from of a joint distribution over a product type.

Similarly to joint distributions over a product type, one might want to build distributions from joint distributions over a triple of distributions. For example, in Table 2, `TripC12.d` permutes the first and the second element, while `Proj13.d` build the marginal w.r.t. the second element. In Sect. 4, we also deal with distributions made of quadruples of distributions but rarely enough so that we do not introduce specific definitions for them.

In the case of multivariate distributions (i.e., distributions of type `{fdist 'rV[A]_n}`), one might want to build the marginals made of the head or the tail, like `Multivar.head_of` or `Multivar.tail_of` in Table 3. In the same table, `Multivar.belast_last` is a distribution over the pairs formed by, on one side, all the elements but the last, and on the other side, the last element. The function `Nth.d` builds a marginal from any index and `Take.d` builds a marginal with a prefix. `PairNth.d` builds a marginal from a distribution over a product type and any index. `PairTake.d` operates a more complicated rearrangement (intuitively, a combination of `Take.d`, `Nth.d`, and `Bivar.snd`). Finally, `PairNth.d` and `PairTake.d` will find theirs use when stating the chain rule for information in Sect. 3.2.3.

In the following, we will need to make explicit mention of the distributions above, whereas they are often implicit in pencil-and-paper proofs.

2.3 Conditional Probability

We use joint distributions to define conditional probabilities.

Given a joint distribution `P : {fdist A * B}` and two events `E : {set A}` and `F : {set B}`, the probability of `E` knowing `F` (i.e., $\Pr[E|F]$) is formalized as follows:

```
Variables (A B : finType) (P : {fdist A * B}).
Definition cPr E F :=
  Pr P (setX E F) / Pr (Bivar.snd P) F.
```

The set `setX E F` is the Cartesian product of the sets `E` and `F`. Using a pencil-and-paper prose that mentions explicitly the underlying distributions (what many textbooks do not do), the definition of `cPr P E F` would read as $\Pr_P[E|F] \stackrel{\text{def}}{=} \frac{\Pr_P(E\times F)}{\Pr_{P_2}(F)}$, where P_2 represents the right marginal of P. This is why we use the notation `\Pr_P[E | F]` for `cPr P E F` in the Coq scripts.

We formalized a number of lemmas about conditional probabilities, such as Bayes' theorem, its general form, etc. We do not provide a complete presentation because their proofs follow known formal proofs [13]. Let us just provide two examples that are used to prove the graphoid axioms. The proofs of most of the graphoid axioms (see Sect. 4) use the product rule (i.e., $\Pr[E \times F|G] = \Pr[E|F \times G]\Pr[F|G]$), which we state formally as follows:

```
Lemma product_rule E F G :
  \Pr_P [setX E F | G] =
    \Pr_(TripA.d P) [E | setX F G] *
    \Pr_(Proj23.d P) [F | G].
```

The proof of intersection uses the conditional property of the universal set, i.e., the fact that $\Pr[U|A] = 1$ where U is the universal set and A is an event with non-zero probability:

```
Variables (A B : finType) (P : {fdist A * B}).
Lemma cPr_1 a : Bivar.snd P a != 0 ->
  \sum_(b in B)
    \Pr_(Swap.d P)[[set b] | [set a]] = 1.
```

In the formal definition, `[set x]` is a MathComp notation for the singleton set $\{x\}$.

3 Application 1: Formalization of Information Theory

As a first application of our formalization of conditional probability and joint distributions, we formalize the basic elements of information theory. This includes lemmas that (to the best of our

Table 1 Distributions using the product type

Original distribution	*Built distribution*	*and its type*
`P : {fdist A * B}`	`Swap.d P`	`{fdist B * A}`
	`Bivar.fst P`	`{fdist A}`
	`Bivar.snd P`	`{fdist B}`

Table 2 Distribution using triples

Original distribution	*Built distribution*	*and its type*
`P : {fdist A * B * C}`	`TripA.d P`	`{fdist A * (B * C)}`
	`TripC12.d P`	`{fdist (B * A) * C}`
	`TripC23.d P`	`{fdist (A * C) * B}`
	`TripC13.d P`	`{fdist (C * B) * A}`
	`Proj13.d P`	`{fdist A * C}`
	`Proj23.d P`	`{fdist B * C}`

Table 3 Distributions using vectors

Original distribution	*Built distribution*	*and its type*
`P : {fdist 'rV[A]_n.+1}`	`Multivar.head_of P`	`{fdist A}`
	`Multivar.tail_of P`	`{fdist 'rV[A]_n}`
	`Multivar.belast_last P`	`{fdist 'rV[A]_n * A}`
`P : {fdist 'rV[A]_n}`	`Nth.d P i`	`{fdist A}`
	`Take.d P i`	`{fdist 'rV[A]_i}`
`P : {fdist 'rV[A]_n * B}`	`PairNth.d P i`	`{fdist A * B}`
`P : {fdist 'rV[A]_n.+1 * B}`	`PairTake.d P`	`{fdist ('rV[A]_i * A) * B}`

knowledge) were not formalized before (e.g., the chain rule for information).

3.1 Entropy and Conditional Entropy

3.1.1 Background: Entropy

Entropy is a measure of the uncertainty of a random variable. In [Sect. 2.1 of [9]], it is defined as follows (for a random variable X with alphabet $\mathcal{X}$ and probability mass function $p(x) \stackrel{\text{def}}{=} \Pr(X = x)$):

$$\mathcal{H}(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x)$$

Since the body of the definition uses only distributions, we formalized it as follows in [6] (Coq notation: `` `H ``):

```
Variables (A : finType) (P : {fdist A}).
Definition entropy :=
  - \sum_(a in A) P a * log (P a).
```

One may wonder what this definition means if $P\,a = 0$ for some a. Here we rely on the fact that Coq assumes all functions (including `log`) to be total. As a result, the product of 0 with `log` 0 is equal to 0, which happens to be what we want here. While Coq uses completed versions of logarithm and division for instance, assuming both of them to be 0 outside of their usual definition domains, theorems on specific functions are restricted to their usual definition domains. For instance $x/x = 1$ is true only if $x \neq 0$, *cf.* lemma `cPr_1` (Sect. 2.3).

3.1.2 Conditional Entropy

When it comes to conditional entropy, the standard textbook [9] gives a choice of several definitions (of course all equivalent) using slightly different notions: a more primitive notion of conditional entropy (which is defined simultaneously), using conditional probabilities, using joint distributions, or using the expectation of a random variable [Equations 2.10–2.13 of [9]]. For example, using conditional probabilities, the definition of conditional entropy reads as follows:

$$\mathcal{H}(Y \mid X) = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x)\Pr[y|x]\log\Pr[y|x]$$

Let us use the definition of conditional probability from Sect. 2.3 to write this definition in Coq. Like for entropy (Sect. 3.1.1), we only care about distributions. We are given a joint distribution `QP : {fdist B * A}` (to be able to write down the conditional probability $\Pr[y|x]$) from which we compute the marginal `P := Bivar.snd QP` (so as to be able to write $p(x)$):

```
(* Module CondEntropy *)
Variables (A B : finType) (QP : {fdist B * A}).
Definition h1 a := - \sum_(b in B)
  \Pr_QP[[set b] | [set a]] *
  log (\Pr_QP[[set b] | [set a]]).
Let P := Bivar.snd QP.
Definition h := \sum_(a in A) P a * h1 a.
```

Let us check the validity of our formal definition with an example of computation and a (non-trivial) property of conditional entropy.

3.1.2.1 Computation of Conditional Entropy

Let us consider the joint distribution with the following probability mass function [Example 2.2.1 of [9]]:

```
(* Module conditional_entropy_example. *)
Definition f : 'I_4 * 'I_4 -> R :=
  [eta (fun=>0) with
 (zero,zero) |-> 1/8,      (zero,one) |-> 1/16,
  (zero,two) |-> 1/16,  (zero,three) |-> 1/4,
  (one,zero) |-> 1/16,      (one,one) |-> 1/8,
   (one,two) |-> 1/16,   (one,three) |-> 0,
  (two,zero) |-> 1/32,      (two,one) |-> 1/32,
   (two,two) |-> 1/16,   (two,three) |-> 0,
(three,zero) |-> 1/32,   (three,one) |-> 1/32,
 (three,two) |-> 1/16, (three,three) |-> 0].
```

Since it is nonnegative and sums to 1, it is a distribution:

```
Lemma f0 : forall x, 0 <= f x.
Proof. ... Qed.
Lemma f1 :
  \sum_(x in {: 'I_4 * 'I_4}) f x = 1.
Proof. ... Qed.
Definition d : {fdist 'I_4 * 'I_4} :=
  FDist.make f0 f1.
```

The constructor `FDist.make` is a variant of the constructor `FDist.mk` of Sect. 2.1. We compute $H(X|Y)$ where X and Y are random variables whose joint distribution is `d`. Using the symbolic computations capabilities of Coq as implemented by tactics such as `lra`, we recover the expected result:

```
Lemma conditional_entropyE :
  CondEntropy.h d = 11/8.
```

3.1.2.2 "Information Can't Hurt"

One important property of information theory is that conditioning reduces entropy, i.e., $H(X|Y) \leq H(X)$ [Thm 2.6.5 of [9]]. Using the formal definitions explained so far, this can be stated in Coq as follows:

```
Variables (A B : finType) (PQ : {fdist A * B}).
Let P := Bivar.fst PQ.
Lemma information_cant_hurt :
  CondEntropy.h PQ <= `H P.
```

We provide a succinct Coq proof in Sect. 3.2.1, using the properties of mutual information.

3.1.3 The Chain Rule for Entropy

The chain rule for entropy is stated for n random variables $X_1, X_2, \ldots, X_n$ drawn according to $p(x_1, x_2, \ldots, x_n)$ [Thm 2.5.1 of [9]]:

$$\mathcal{H}(X_1,\ldots,X_n) = \sum_{i=1}^{n}\mathcal{H}(X_i \mid X_{i-1},\ldots,X_1)$$

First, we observe that $0 < n$. We reflect this in the formal statement below by considering `n.+1` instead of `n`. Second, we observe that the pencil-and-paper notation is such that $\mathcal{H}(X_i \mid X_{i-1},\ldots,X_1) = \mathcal{H}(X_1)$ when $i = 1$. This is why, in the formal statement below, we distinguish the case where $i = 1$ from the case where $1 < i$ (which become `i = 0` and `0 < i` in Coq where we count from `0`):

```
Lemma chain_rule_rV
  A n (P : {fdist 'rV[A]_n.+1}) :
`H P = \sum_(i < n.+1)
  if i == 0 then
```

```
 `H (Multivar.head_of P)
else
 CondEntropy.h (Swap.d (Multivar.belast_last
   (Take.d P (lift ord0 i)))).
```

In the case where `i = 0`, the chain rule returns `` `H (Multivar.head_of P)``, which is equal to `` `H P`` since `P` is a vector with only one element. The case with `0 < i` is a little bit involved. The expression `lift ord0 i` means $i+1$; it is written with `lift` so that it has the appropriate type `'I_n.+2`, the type of natural number strictly less than `n.+2` [18]. The calls to `Take.d`, `Multivar.belast_last`, and `Swap.d` (see Tables 1 and 3) build the product distribution corresponding to $(X_{i+1}, (X_i, \ldots, X_1))$. This may look a little bit more complicated than what one could expect with lists (instead of (finite-size) vectors), but since distributions have finite support we cannot easily recast the problem in terms of "distributions of lists".

The proof is by induction on n and uses as an intermediate lemma the "chain rule" [Thm 2.2.1 of [9]], which is actually a version of the chain rule for entropy but restricted to three random variables. One can find this restricted version in related work (e.g., [20]), but we are not aware of an existing generalization to n distributions. The formal statement above shows that the need to build a variety of joint distributions makes the generalization not immediate. It is nevertheless useful, for example to prove the chain rule for information (see Sect. 3. 2. 3) or Han's inequality (see [7]).

3. 2 Mutual Information and Conditional Mutual Information

In this section, we give an overview of our formalization of mutual information, conditional mutual information, and the chain rule for information. It is technically a bit more involved than the formalization of conditional entropy in Sect. 3. 1, but it uses similar ideas, hence the shorter presentation.

3. 2. 1 Mutual Information

Mutual information quantifies the information shared by two random variables. Given two random variables X and Y with joint probability mass function $p(x,y)$ and marginals $p(x)$ and $p(y)$, mutual information is defined as $\mathcal{I}(X\,;\,Y) = \mathcal{D}(p(x,y)\,||\,p(x)p(y))$ where $\mathcal{D}(\cdot\,||\,\cdot)$ is the Kullback-Leibler distance [Sect. 2.3 of [9]] (also called *relative entropy*). The formal definition is immediate given that we have already defined in previous work [6] the relative entropy `D(... || ...)` and the product distribution `` `x``:

```
Variables (A B : finType) (PQ : {fdist A * B}).
Let P := Bivar.fst PQ.
Let Q := Bivar.snd PQ.
Definition mi := D(PQ || P `x Q).
```

The notation ``P `x Q`` is for the distribution with probability mass function $(x,y) \mapsto P(x) \times Q(x)$. It is defined using a more generic construct (where the second projection may depend on the first one) that will be better explained in the context of Sect. 3. 3.

This formal definition is equipped with the mandatory lemmas that, among others, provide a short proof for the "information can't hurt" property of Sect. 3. 1. 2:

```
Lemma information_cant_hurt :
  CondEntropy.h PQ <= `H P.
Proof.
rewrite -subR_ge0 -MutualInfo.miE;
  exact: MutualInfo.mi_ge0.
Qed.
```

`MutualInfo.miE` provides an alternative definition of mutual information in terms of entropy (i.e., $\mathcal{I}(X\,;\,Y) = \mathcal{H}(X) - \mathcal{H}(X\,|\,Y)$ [Thm 2.4.1 of [9]]) and `MutualInfo.mi_ge0` establishes that mutual information is nonnegative [p. 28 of [9]].

3. 2. 2 Conditional Mutual Information

Conditional mutual information quantifies the information shared by two random variables *under the assumption that the result of a third random variable is already known*: $\mathcal{I}(X;Y|Z) = \mathcal{H}(X\,|\,Z) - \mathcal{H}(X\,|\,Y,Z)$ [Sect. 2.5 of [9]]. The formalization is immediate given the definition of conditional entropy from Sect. 3.1.2:

```
Variables (A B C : finType)
          (PQR : {fdist A * B * C}).
Definition cmi :=
  CondEntropy.h (Proj13.d PQR) -
  CondEntropy.h (PairA.d PQR).
```

3.2.3 The Chain Rule for Information

Mutual information and conditional mutual information satisfy a chain rule [Thm 2.5.2 of [9]]:

$$\mathcal{I}(X_1, X_2, \ldots, X_n ; Y) = \sum_{i=1}^{n} \mathcal{I}(X_i; Y | X_{i-1}, \ldots, X_1)$$

It looks similar to the chain rule for entropy and conditional entropy of Sect. 3.1.3.

For the formal statement, we first prepare the joint distribution $X_1, X_2, \ldots, X_n, Y$ as the distribution `PY` of type `{fdist 'rV[A]_n.+1 * B}`:

```
Variables (A : finType). Let B := A.
Variables (n : nat)
          (PY : {fdist 'rV[A]_n.+1 * B}).
```

For the case where $i = 1$, we take the mutual information of the distribution X_1, Y, i.e., `(PairNth.d PY ord0)` (`ord0` is the natural number 0 but with the type `'I_n.+1`). For the case where $1 < i$, we take the conditional mutual information of the distribution $X_i, Y, (X_{i-1}, \ldots, X_1)$, i.e., the distribution `TripC23.d (TripC12.d (PairTake.d PY i))` that we build as the function `f23`:

```
Let f (i : 'I_n.+1) : {fdist A * 'rV[A]_i * B} :=
  TripC12.d (PairTake.d PY i).
Let f23 (i : 'I_n.+1) : {fdist A * B * 'rV[A]_i} :=
  TripC23.d (f i).

Lemma chain_rule_information :
  MutualInfo.mi PY = \sum_(i < n.+1)
    if i == 0 then
      MutualInfo.mi (PairNth.d PY ord0)
    else
      cmi (f23 i).
```

The proof uses the chain rule for entropy of Sect. 3.1.3.

3.3 Convexity of Entropy and Mutual Information

The goals of this section are formal proofs of convexity for entropy and mutual information. We start by defining convexity using the notion of *convex space* [15].

Let `A` and `B` be convex spaces. A function `f` of type `A -> B` is convex when it satisfies the predicate `convex_function`:

```
Definition convex_function_at a b (p : prob) :=
  f (a <| p |> b) <= f a <| p |> f b.
Definition convex_function :=
  forall a b (p : prob),
    convex_function_at f a b p.
```

The type `prob` is for reals between 0 and 1; `a <| p |> b` is a notation for $p \times a + (1-p) \times b$. Similarly, the predicate `concave_function` is for functions f such that $-f$ is convex. The above definition applies to the type of finite distributions and to the type of Coq reals `R` which both happen to be convex spaces; in particular, to indicate to the type system of Coq that finite distributions over `A` are to be seen as forming a convex space, we use the type `fdist_convType A` instead of `{fdist A}`. We do not explain here in details the formalization of convex spaces, it is the object of a forthcoming publication, in the meantime all the details are in the Coq formalization [file `convex_choice.v` of [7]].

Given the above definitions, we can state the convexity of entropy as follows [Thm 2.7.3 of [9]]:

```
Variable (A : finType).
Hypothesis A_not_empty : 0 < #|A|.
Lemma entropy_concave : concave_function
  (fun P : fdist_convType A => `H P).
```

The proof relies on a convexity property of relative entropy.

The convexity of mutual information requires an additional construct. Let us first recall the pencil-and-paper statement. Let (X, Y) be two random variables drawn according to $p(x, y) = p(x)\Pr[y|x]$. For fixed $\Pr[y|x]$, $\mathcal{I}(X ; Y)$ is a concave function of $p(x)$ [first part of Thm 2.7.4 of [9]]. In the formal statement, we represent $\Pr[y|x]$ as `W : A -> {fdist B}` (i.e., a stochastic matrix), and $p(x)$ is given as the parameter `P : {fdist A}`. We build the joint distribution $p(x, y)$ using the function `make_joint` defined as follows:

```
(* Module CFDist *)
```

```
Variables (A B : finType).
Record t :=
  mkt {P : {fdist A} ; W :> A -> {fdist B}}.
Definition joint_of (x : t) : {fdist A * B} :=
  ProdDist.d (P x) (W x).
Definition make_joint (P : dist A)
  (W : A -> dist B) : {fdist A * B} :=
  joint_of (mkt P W).
```

Given a distribution `P` and a stochastic matrix `W`, `ProdDist.d` builds the "product distribution" with probability mass function $(x, y) \mapsto P(x) \times Q(x, y)$. (This explains the product distribution of Sect. 3.2.1: `P `x Q` is actually a notation for `ProdDist.d P (fun=> Q)`.) By construction, the left marginal of the joint distribution is `P`, which is one part of the original condition (specifically the condition $p(x, y) = p(x) \Pr[y|x]$) and lets us prove the concavity of mutual information (when its second argument is blocked):

```
Variables (A B : finType) (Q : A -> {fdist B}).
Hypothesis B_not_empty : 0 < #|B|.
Lemma mutual_information_concave :
  concave_function
  (fun P : fdist_convType A =>
    MutualInfo.mi (CFDist.make_joint P Q)).
```

The proof uses the convexity of entropy as an intermediate step.

The convexity of mutual information (as a function of $\Pr[y|x]$ for fixed $p(x)$) [second part of Thm 2.7.4 of [9]] follows along similar lines (see [7]).

3.4 More Information Theory

We were actually able to formalize more information theory than what we have explained so far. At the time of this writing, we have almost completed the formalization of [Chapter 2 of [9]]: only the sections about sufficient statistics and Fano's inequality are left unexplored.

Among theorems that we have not explained are also the chain rule for relative entropy [Thm 2.5.3 of [9]], the independence bound on entropy [Thm 2.6.6 of [9]], the definition of Markov chains, and the data-processing inequality [Thm 2.8.1 of [9]].

This work uses and complements our previous work that already provides in Coq several theorems from [Chapter 2 of [9]]: Jensen's inequality [Thm 2.6.2 of [9]] (see [5]), the log sum inequality [Thm 2.7.1 of [9]] (see [6]), and the proof that the nonnegativity of the second derivative implies convexity [Thm 2.6.1 of [9]].

4 Application 2: Formalization of Conditional Independence

As a second application of our formalization of conditional probability and joint distributions, we formalize conditional independence and we validate this formal definition by establishing the graphoid axioms.

4.1 Random Variables

Given a distribution `P : {fdist U}`, a random variable over `A` is defined as a function of type `U -> A`, where `A` is expected to be some finite type[†1] for the values that the random variable can take:

```
Definition RV {U : finType} (P : {fdist U})
  (A : eqType) := U -> A.
```

This is the same definition as Moreira [20]. In the following, we use `{RV P -> A}` as a notation for `RV P A`.

A random variable `X : {RV P -> A}` induces a distribution over its codomain. This is the distribution whose probability mass function associates to each element `a` of `A` the probability of the event corresponding to the pre-image of `a` via `X`, i.e., `FDistMap.d X P` (see Sect. 2.2.1). We introduce the Coq notation `\Pr[X = a]` corresponding to the probability `Pr (FDistMap.d X P) [set a]`, and more generally the Coq notation `\Pr[X = a | Y = b]` for `\Pr_(FDistMap.d [% X, Y] P)[[set a] | [set b]]`. Recall the notation for conditional probability from Sect. 2.3. The complete Coq definitions and proofs are available online [files `proba.v` and `cinde.v` of

†1 This is not enforced by the definition that only requires a type with a decidable equality but this is required by most useful lemmas.

[7]].

In this setting, we observe in particular that we can create random variables by pairing existing random variables:

```
Variables (U : finType) (P : {fdist U}).
Variables (A : finType) (X : {RV P -> A})
          (B : finType) (Y : {RV P -> B}).
Definition RV2 : {RV P -> A * B} :=
  fun x => (X x, Y x).
```

In the following, `[% X, Y, ..., Z]` denotes the iterated pairing of the random variables `X`, `Y`, ..., `Z`.

4. 2 Conditional Independence

We are concerned with the formalization of the predicate $X \perp Y \,|\, Z$, which intuitively means that X is independent of Y given Z. We first recall the textbook definition.

Definition 1 (Conditional Independence [Definition 2.4 of [16]]). *Let X, Y, and Z be random variables and P be a distribution. X is conditionally independent of Y given Z in the distribution P if for all values a, b, and c (belonging resp. to the codomains of X, Y, and Z), we have:*

$$\Pr(X = a, Y = b | Z = c) = \Pr(X = a | Z = c) \Pr(Y = b | Z = c).$$

We now move on to the formal definition of conditional independence. First, we give ourselves a sample space `U` with a distribution `P : {fdist U}`:

```
Variables (U : finType) (P : {fdist U}).
```

Second, we declare three random variables `X`, `Y`, and `Z`:

```
Variables (A B C : finType).
Variables (X : {RV P -> A}) (Y : {RV P -> B})
  (Z : {RV P -> C}).
```

Finally, we state the property of conditional independence (Coq notation: `X _|_ Y | Z`):

```
Definition cinde_drv := forall a b c,
  \Pr[ [% X, Y] = (a, b) | Z = c ] =
    \Pr[ X = a | Z = c ] * \Pr[ Y = b | Z = c].
```

Comparison with the pencil-and-paper definition is immediate. The fact that each probability appearing in this definition relies on different distributions can still be guessed by the reader from the occurrences of the random variables `X`, `Y`, `Z`, but moreover it is now possible to unfold formal definitions to discover explicitly that, say, distributions in the right hand-side are marginals of the distribution in the left hand-side. One goal of our formalization is however to provide enough definitions and lemmas to avoid doing unfolding and keeping formal proofs as simple as possible.

4. 3 Conditional Independence is a Graphoid

Conditional independence satisfies the graphoid axioms [21]:

- Symmetry: $X \perp Y \,|\, Z$ implies $Y \perp X \,|\, Z$.
- Decomposition: $X \perp Y, W \,|\, Z$ implies $X \perp Y \,|\, Z$.
- Weak union: $X \perp Y, W \,|\, Z$ implies $X \perp Y \,|\, Z, W$.
- Contraction: $X \perp W \,|\, Z, Y$ and $X \perp Y \,|\, Z$ imply $X \perp Y, W \,|\, Z$.
- Intersection: $X \perp Y \,|\, Z, W$ and $X \perp W \,|\, Z, Y$ imply $X \perp Y, W \,|\, Z$ for positive distributions.

These axioms have an intuitive interpretation. For example, symmetry means that in a state Z where X says nothing new about Y, then Y says nothing new about X; decomposition means that if Y, W say nothing new about X, then Y alone does not say anything new about X; etc.

We have proved that the graphoid axioms for conditional independence are satisfied by the construction of finite discrete probability theory on top of finite types and real numbers seen in Sect. 2. 1. We detail the formal proofs below, pencil-and-paper proofs have been reconstructed from [16] [22]. The full script is available online [file `cinde.v` of [7]].

4. 3. 1 The Proof of Symmetry in Coq

Let us recall the pencil-and-paper proof:

Lemma 1 (Symmetry). *$X \perp Y \,|\, Z$ implies $Y \perp X \,|\, Z$.*

Proof.

$$\begin{aligned} P(Y,X|Z) &= P(X,Y|Z) \\ &\qquad \text{(by set-theoretic reasoning)} \\ &= P(X|Z)P(Y|Z) \\ &\qquad \text{(because } X \perp Y \mid Z\text{)} \\ &= P(Y|Z)P(X|Z) \\ &\qquad \text{(by commutativity of multiplication)} \end{aligned}$$

□

In Coq, we write the statement "$X \perp Y \mid Z$ implies $Y \perp X \mid Z$" as follows:

```
Variable (U : finType) (P : {fdist U}).
Variables (A B C : finType) (X : {RV P -> A})
  (Y : {RV P -> B}) (Z : {RV P -> C}).
Lemma symmetry : X _|_ Y | Z -> Y _|_ X | Z.
```

The proof script follows the pencil-and-paper proof:

```
1 Lemma symmetry : X _|_ Y | Z -> Y _|_ X | Z.
2 Proof.
3 move=> H b a c.
4 rewrite RV_Pr_lC.
5 rewrite H.
6 by rewrite mulRC.
7 Qed.
```

At the beginning of the proof, there is no syntactic match between the random variables in the hypothesis and in the goal: the hypothesis is about the random variables `[% X, Y, Z]` whereas the goal is about `[% Y, X, Z]`. The purpose of line 4 is to swap the two leading random variables so as to be able to rewrite with the hypothesis at line 5. We apply the commutativity of multiplication at line 6, which completes the proof.

4. 3. 2 The Proof of Decomposition in Coq

The proof of decomposition is more involved than symmetry; it relies in particular on "reasoning by cases", i.e., $P(X|Y) = \sum_z P(X,z|Y)$, which becomes in our formalization:

```
Lemma reasoning_by_cases E F :
  \Pr[ X \in E | Y \in F ] =
  \sum_(z <- fin_img Z)
    \Pr[ [% X, Z] \in setX E [set z] | Y \in F ].
```

where `E` and `F` are two events and `fin_img` is the (finite) image of a random variable. The notation `\Pr[X \in E | Y \in F]` stands for

```
\Pr_(FDistMap.d [% X, Y] P)[E | F]
```

and comes as a slight generalization of the notation introduced in Sect. 4. 1.

Lemma 2 (Decomposition)**.** $X \perp Y, W \mid Z$ *implies* $X \perp Y \mid Z$.

Proof. [Sect. 2.1.4.3 of [16]]

$$\begin{aligned} P(X,Y|Z) &= \sum_w P(X,Y,w|Z) \\ &\qquad \text{(reasoning by cases)} \\ &= \sum_w P(X|Z)P(Y,w|Z) \\ &\qquad \text{(by } X \perp Y, W \mid Z\text{)} \\ &= P(X|Z)\sum_w P(Y,w|Z) \\ &\qquad \text{(by distributivity)} \\ &= P(X|Z)P(Y|Z) \\ &\qquad \text{(reasoning by cases)} \end{aligned}$$

□

Here follows the proof statement in Coq:

```
Variables (U : finType) (P : {fdist U})
  (A B C D : finType).
Variables (X : {RV P -> A}) (Y : {RV P -> B})
  (Z : {RV P -> C}) (W : {RV P -> D}).
Lemma decomposition :
  X _|_ [% Y, W] | Z -> X _|_ Y | Z.
```

Figure 1 displays a proof script that proved decomposition. Although it is not necessary (and actually lengthen the proof artificially), we have added `transitivity` steps to easy comparison with the pencil-and-paper proof. The first subgoal appears explicitly at line 4. Solving it is essentially a matter of reasoning by cases (see line 5). The second subgoal appears explicitly at line 7. It is essentially solved by using the hypothesis of conditional independence at line 9. Line 10 is where we use the distributivity of multiplication w.r.t. addition. Line 11 is the conclusive reasoning by cases. The lines that we have not explained in details are just trivial manipulations of random variables and finites sets. For example, `RV_Pr_lA` is a lemma to rearrange random variables appearing in conditional probabilities by appealing to their associativity.

```
Lemma decomposition : X _|_ [% Y, W] | Z -> X _|_ Y | Z.
Proof.
move=> H a b c.
transitivity (\sum_(d <- fin_img W) \Pr[ [% X, [% Y, W]] = (a, (b, d)) | Z = c]).
  rewrite (reasoning_by_cases W); apply eq_bigr => /= d _.
  by rewrite RV_Pr_lA setX1.
transitivity (\sum_(d <- fin_img W)
  \Pr[ X = a | Z = c] * \Pr[ [% Y, W] = (b, d) | Z = c]).
  by apply eq_bigr => d _; rewrite H.
rewrite -big_distrr /=; congr (_ * _).
rewrite (reasoning_by_cases W); apply eq_bigr => d _.
by rewrite setX1.
Qed.
```

Figure 1 Proof script for the lemma `decomposition`

4. 3. 3 The Proof of Weak Union in Coq

The pencil-and-paper proof of weak union is shorter than decomposition, but uses it as an intermediate step.

Lemma 3 (Weak Union)**.** $X \perp Y, W \mid Z$ *implies* $X \perp Y \mid Z, W$.

Proof.

$$
\begin{aligned}
P(X,Y|Z,W) &= P(X|Y,Z,W)P(Y|Z,W) \\
&\qquad\text{(by the product rule)} \\
&= P(X|Z)P(Y|Z,W) \\
&\qquad\text{(because } X \perp Y, W \mid Z\text{)} \\
&= P(X|Z,W)P(Y|Z,W)
\end{aligned}
$$

The last step is because $X \perp W \mid Z$ by decomposition (Lemma 2) from $X \perp Y, W \mid Z$. □

Here follows the proof statement in Coq:

```
Variables (U : finType) (P : {fdist U})
  (A B C D : finType).
Variables (X : {RV P -> A}) (Y : {RV P -> B})
  (Z : {RV P -> C}) (W : {RV P -> D}).

Lemma weak_union :
  X _|_ [% Y, W] | Z -> X _|_ Y | [% Z, W].
```

See Fig. 2 for the complete proof script. The first step appears at line 4 and is completed by application of the product rule at line 6. Note that this is a variant of the product rule presented in Sect. 2. 3: it is specialized to random variables. The second step appears at line 7 and is completed by using the conditional independence hypothesis at line 11. The use of decomposition can be observed at line 14 (where the lemma `cinde_drv_2C` just operates a reordering of random variables) and is followed by another use of the conditional independence hypothesis at line 15.

The proof script is maybe longer than what one could expect given the previous examples. This is because of corner cases with null probabilities, that are often ignored in pencil-and-paper proofs. Here they come from the application of the lemma `cindeP`, which establishes

$$X \perp Y \mid Z \to P(X|Y,Z) = P(X|Z)$$

under the hypothesis that $P(Y,Z) \neq 0$.

4. 3. 4 The Proof of Contraction in Coq

The proof of contraction uses that same ideas as previous proofs:

Lemma 4 (Contraction)**.** $X \perp W \mid Z, Y$ *and* $X \perp Y \mid Z$ *imply* $X \perp Y, W \mid Z$.

Proof.

$$
\begin{aligned}
P(X,Y,W|Z) &= P(X|Y,W,Z)P(Y,W|Z) \\
&\qquad\text{(by the product rule)} \\
&= P(X|Y,Z)P(Y,W|Z) \\
&\qquad\text{(because } X \perp W \mid Z, Y\text{)} \\
&= P(X|Z)P(Y,W|Z) \\
&\qquad\text{(because } X \perp Y \mid Z\text{)}
\end{aligned}
$$

□

We display the formal proofs of contraction in Fig. 3 for the sake of completeness but without further comments since it uses the same ingredients as

```
Lemma weak_union : X _|_ [% Y, W] | Z -> X _|_ Y | [% Z, W].
Proof.
move=> H a b [c d].
transitivity (\Pr[ X = a | [% Y, Z, W] = (b, c, d)] *
  \Pr[ Y = b | [% Z, W] = (c, d)]).
  by rewrite RV_product_rule RV_Pr_rA.
transitivity (\Pr[ X = a | Z = c] * \Pr[ Y = b | [% Z, W] = (c, d)]).
  rewrite RV_Pr_rAC.
  case/boolP : (\Pr[ [% Y, W, Z] = (b, d, c)] == 0) => [/eqP|] H0.
  - by rewrite [X in _ * X = _ * X]RV_cPrE RV_Pr_A RV_Pr_AC H0 div0R !mulR0.
  - by rewrite (cindeP _ H).
case/boolP : (\Pr[ [% Z, W] = (c, d) ] == 0) => [/eqP|] ?.
- by rewrite [X in _ * X = _ * X]RV_cPrE RV_Pr_domin_snd ?(div0R,mulR0).
- have {H}H : X _|_ W | Z by move/cinde_drv_2C : H; apply decomposition.
  by rewrite [in X in _ = X * _]RV_Pr_rC (cindeP _ H) // RV_Pr_C.
Qed.
```

Figure 2 Proof script for the lemma weak_union

Fig. 2.

4.3.5 The Proof of Intersection in Coq

The proof of intersection is the most involved of all graphoid axioms. In particular, intersection uses contraction, and reasoning by cases and the product rule as intermediate steps.

Lemma 5 (Intersection). *$X \perp Y \mid Z, W$ (1) and $X \perp W \mid Z, Y$ (2) imply $X \perp Y, W \mid Z$ provided that the distribution Y, Z, W is positive (3) and the type of the codomain of W is not empty (4).*

Proof. It suffices to show $X \perp Y \mid Z$ by contraction (Lemma 4) because we already have (2).

The goal is therefore $P(X, Y|Z) = P(X|Z)P(Y|Z)$ which is proved as follows:

$$\begin{aligned} P(X|Z)P(Y|Z) &= \sum_w P(X, w, Z)P(Y|Z) \\ &\qquad \text{(reasoning by cases)} \\ &= \sum_w P(X, Y|Z)P(w|Z) \\ &\qquad \text{(see (*) below)} \\ &= P(X, Y|Z)\sum_w P(w|Z) \\ &\qquad \text{(by distributivity)} \\ &= P(X, Y|Z) \end{aligned}$$

The last step is by conditional probability of the universal set (Sect. 2.3) and (4).

Proof of step (*): By (2), (3), and the product rule, we have $P(X|Y, Z) = P(X|W, Z, Y)$.

By (1), (3), and the product rule, we have $P(X|W, Z) = P(X|Y, W, Z)$.

Therefore we have $P(X|Y, Z) = P(X|W, Z)$.

By (3) and the product rule, we derive

$$\frac{P(X, Y|Z)}{P(Y|Z)} = \frac{P(X, W|Z)}{P(W|Z)},$$

which implies

$$P(X, Y|Z)P(W|Z) = P(X, W|Z)P(Y|Z)$$

and

$$\sum_w P(X, Y|Z)P(w|Z) = \sum_w P(X, w|Z)P(Y|Z).$$

□

The longer proof of intersection naturally translated into a longer proof script that we partially display in Fig. 4. Its structure is however still readable. The first step using contraction appears at line 4. It is a "it suffices to" step that is conveniently represented by the `suff` tactic of SSReflect. The other steps of the main proof appear respectively at line 6 (reasoning by cases), line 10 ((*) step, lines 9 and 27 (distributivity), and after line 27 for the conclusion. The technicalities of the (*) step have been omitted but can be found online [file `cinde.v` of [7]].

```
Lemma contraction : X _|_ W | [% Z, Y] -> X _|_ Y | Z -> X _|_ [% Y, W] | Z.
Proof.
move=> H1 H2 a [b d] c.
rewrite RV_product_rule.
transitivity (\Pr[X = a | [% Y, Z] = (b, c)] * \Pr[[% Y, W] = (b, d) | Z = c]).
  rewrite -RV_Pr_rA [in X in X * _ = _]RV_Pr_rC -RV_Pr_rA.
  case/boolP : (\Pr[ [% W, [% Z, Y]] = (d, (c, b))] == 0) => [/eqP|] H0.
    rewrite [in X in _ * X = _ * X]RV_cPrE.
    by rewrite -RV_Pr_A RV_Pr_C -RV_Pr_A H0 div0R !mulR0.
  by rewrite (cindeP _ H1) // RV_Pr_rC.
case/boolP : (\Pr[ [% Y, Z] = (b, c) ] == 0) => [/eqP|] H0.
- rewrite [X in _ * X = _ * X]RV_cPrE.
  by rewrite RV_Pr_AC RV_Pr_domin_fst ?div0R ?mulR0.
- by rewrite (cindeP _ H2).
Qed.
```

Figure 3 Proof script for the lemma `contraction`

4.4 Application of the Graphoid Axioms

Now that we have proved the graphoid axioms, we can derive most rules to reason formally about probabilistic models. For example, the *chaining rule* uses *all* the *semi-graphoid axioms* (i.e., the graphoid axioms except intersection):

```
Lemma chaining_rule :
  X _|_ Z | Y /\ [% X, Y] _|_ W | Z ->
  X _|_ W | Y.
Proof.
case=> ? ?.
suff : X _|_ [% W, Z] | Y by move/decomposition.
apply/cinde_drv_2C/contraction => //.
exact/cinde_drv_3C/symmetry/weak_union/symmetry.
Qed.
```

The lemmas `cinde_drv_XC` operate mere reorderings of random variables (we already saw the lemma `cinde_drv_2C` in Sect. 4.3.3).

See also the *mixing rule* for another example of derived rule [file `cinde.v` of [7]].

5 Related Work

The theory of conditional probabilities (Bayes' theorems, etc.) is developed in HOL and applied to the analysis of the binary asymmetric channel [13], but there does not seem to be any theory of joint distributions in this work. We were able to reproduce the same theory of conditional probabilities as [13] in the setting of Sect. 2.3, except for lemmas involving countable unions which are not handled by the theory of finite sets of MATHCOMP. Information-theoretic notions such as entropy, relative entropy, and mutual information are defined in HOL using a formalization of probability based on measure theory and Lebesgue integration [19], but there does not seem to be much lemmas about them. There are more information-theoretic notions (e.g., conditional mutual information information) defined Isabelle/HOL using Lebesgue integration [14]. Compared to the work above, our work features much more lemmas (all the chain rules, lemmas about conditional independence, etc.).

Our work uses a COQ library [7] that comes with a formalization of distributions and probabilities (including basic lemmas such as the weak law of large numbers) and applications to information theory. This work already features definitions of conditional entropy and mutual information but specialized for a setting with one input distribution P and one stochastic matrix W, which models a channel of communication. In this setting, one starts by defining a joint distribution $J(P, W)$ and then uses this distribution to define conditional entropy $H(W|P)$ and mutual information $I(P, W)$ (these notations departs from [9] but are not uncommon, see, e.g., [10]). We can prove that each entry of W corre-

```
Lemma intersection : X _|_ Y | [% Z, W] -> X _|_ W | [% Z, Y] -> X _|_ [% Y, W] | Z.
Proof.
move=> H1 H2.
suff : X _|_ Y | Z by apply contraction.
move=> a b c; apply/esym.
rewrite [in X in X * _ = _](reasoning_by_cases W).
evar (h : D -> R); rewrite (eq_bigr h); last first.
  move=> d _; rewrite setX1 /h; reflexivity.
rewrite {}/h big_distrl /=.
have <- : \sum_(d <- fin_img W)
            \Pr[ [% X, Y] = (a, b) | Z = c] * \Pr[ W = d | Z = c] =
          \sum_(d <- fin_img W)
            \Pr[ [% X, W] = (a, d) | Z = c] * \Pr[ Y = b | Z = c].
  suff H : forall d, \Pr[ [% X, Y] = (a, b) | Z = c] / \Pr[ Y = b | Z = c ] =
                 \Pr[ [% X, W] = (a, d) | Z = c] / \Pr[ W = d | Z = c ].
    ... (* by using properties of positive distributions *)
  suff H : forall d, \Pr[ X = a | [% Y, Z] = (b, c)] =
                 \Pr[ X = a | [% W, Z] = (d, c)].
    ... (* by using the product rule *)
  have {H2}H2 : forall d, \Pr[ X = a | [% Y, Z] = (b, c)] =
                     \Pr[ X = a | [% W, Z, Y] = (d, c, b)].
    ... (* by using the product rule and the second conditional independence hypothesis *)
  have {H1}H1 : forall d, \Pr[ X = a | [% W, Z] = (d, c)] =
                     \Pr[ X = a | [% Y, W, Z] = (b, d, c)].
    ... (* by using the product rule and the first conditional independence hypothesis *)
  by move=> d; rewrite {H2}(H2 d) {}H1 RV_Pr_rC RV_Pr_rA.
rewrite -big_distrr /= cPr_1_RV ?mulR1 //.
move: (P0 b c D_not_empty); apply: contra.
by rewrite RV_Pr_AC => /eqP/(RV_Pr_domin_snd [% Y, W] (b, D_not_empty)) ->.
Qed.
```

Figure 4 Partial proof script for the lemma `intersection` (see [file `cinde.v` of [7]] for the complete proof script)

sponds to a conditional probability using $J(P, W)$:

```
Variables (A B : finType) (W : `Ch_1(A, B))
  (P : {fdist A}).
Let QP := Swap.d (`J(P, W)).
Lemma WcPr : forall a b, P a != 0 ->
  W a b = \Pr_QP[[set b] | [set a]].
```

It follows that $H(W|P)$ is equal to `CondEntropy.h QP` and $I(P, W)$ is equal to `MutualInfo.mi QP`. Similarly, in [2], we defined *aposteriori probabilities* using P and W:

$$P^W(x|y) \stackrel{\text{def}}{=} \frac{P(x)W^n(y|x)}{\sum_{x' \in A^n} P(x')W^n(y|x')}.$$

We can now show that $P^W(x|y)$ is equal to `\Pr_(`J(P, W ``^ n))[[set x]|[set y]]` where `` `J(P, W ``^ n) `` is the joint distribution of the n^{th} extension of the channel W. See [file `chap2.v` of [7]] for details. In other words, the work we present in this paper simplifies and generalizes our previous work.

There is another formalization of information theory that uses a similar setting [20]. Distributions are defined similarly, which is not surprising because it is natural to use the big operators of MathComp in this way. The definition of random variable in Sect. 4.1 is the same definition as [20]. Yet, that work stops at an early stage with the chain rule for entropy restricted to three variables and the definition of mutual information. Another limitation of this formalization is that it axiomatizes the logarithm, whereas InfoTheo is compatible with the logarithm defined in the standard library of Coq.

Our work connects concretely to the semantics of probabilistic programming languages. We are currently using the InfoTheo library to provide a formal model for a monad combining probability and nondeterminism [3]. It happens that such a model requires the notion of convex spaces that we have developed in Sect. 3.3.

Regarding conditional independence, one can find a formal definition of the graphoid axioms in Coq/SSReflect [23] but this work does not seem to provide the underlying formalization of probabilities; instead, it focuses on a high-level algebraic presentation and its application.

The formalization of conditional independence can be tackled in a different way using *string diagrams* [Proposition 6.10 of [8]].

6 Conclusion

In this paper, we proposed a formalization of conditional probabilities and joint distributions validated by two original applications: a formalization of the foundations of information theory and a formalization of conditional independence. Our formalization of information theory extends previous work with lemmas that were not formalized before (e.g., the chain rule for information, convexity of entropy). Our second application is (to the best of our knowledge) the first formal account *from the ground up* of the graphoid axioms of conditional independence. The complete formalization is available online as a conservative extension of the InfoTheo library [7] (Table 4 summarizes the relevant files).

Using our work, it is now possible to address new challenges in formalization of information theory and artificial intelligence. Next, we plan to tackle the formalization of Fano's inequality, the formalization of Bayesian networks, and more generally formal reasoning about probabilistic graphical models.

Acknowledgments The authors are thankful to Kenta Cho, Shin-ya Katsumata, and Keisuke Nakano for fruitful discussions about the work presented in this paper, to Paul Fournier for his proofreading, and to the anonymous reviewers of PPL2019 for their helpful comments and for suggesting the application to Han's inequality. The authors acknowledge partial support from a Grant-in-Aid for Scientific Research (B) (project number 18H03204).

References

[1] R. Affeldt, C. Cohen, and D. Rouhling. Formalization techniques for asymptotic reasoning in classical analysis. *Journal of Formalized Reasoning*, 11(1):43–76, 2018.

[2] R. Affeldt and J. Garrigue. Formalization of error-correcting codes: from Hamming to modern coding theory. In *6th Conference on Interactive Theorem Proving (ITP 2015), Nanjing, China, August 24–27, 2015*, volume 9236 of *Lecture Notes in Computer Science*, pages 17–33. Springer, Aug 2015.

[3] R. Affeldt, J. Garrigue, D. Nowak, and T. Saikawa. Monadic equational reasoning in Coq. `https://github.com/affeldt-aist/monae` (last access: 2019/09/29). Main reference: [4].

[4] R. Affeldt, D. Nowak, and T. Saikawa. A Hierarchy of Monadic Effects for Program Verification using Equational Reasoning. In *13th International Conference on Mathematics of Program Construction (MPC 2019), Porto, Portugal, October 7–9, 2019*, volume 11825 of *Lecture Notes in Computer Science*. Springer, Oct 2019.

[5] R. Affeldt, J. Garrigue, and T. Saikawa. Examples of formal proofs about data compression. In *International Symposium on Information Theory and Its Applications (ISITA 2018), Singapore, October 28–31, 2018*, pages 665–669. IEICE. IEEE Xplore, Oct 2018.

[6] R. Affeldt, M. Hagiwara, and J. Sénizergues. Formalization of Shannon's theorems. *Journal of Automated Reasoning*, 53(1):63–103, 2014.

[7] R. Affeldt, M. Hagiwara, J. Sénizergues, J. Garrigue, K. Sakaguchi, T. Asai, T. Saikawa, and N. Obata. A Coq formalization of information theory and linear error-correcting codes. `https://github.com/affeldt-aist/infotheo` (last access: 2019/09/29).

[8] K. Cho and B. Jacobs. Disintegration and bayesian inversion via string diagrams. *Mathematical Structures in Computer Science*, 2019.

[9] T. M. Cover and J. A. Thomas. *Elements of information theory*. Wiley, 2006. 2nd edition.

[10] I. Csiszár and J. Körner. *Information Theory: Coding Theorems for Discrete Memoryless*

Table 4 Overview of the formalization in this paper

File	Contents	Sections in this paper
InfoTheo *files that required extensions for this paper*		
`probability/proba.v`	Distributions and probabilities	Sect. 2. 1
	Joint distributions	Sect. 2. 2
New files that come with this paper		
`probability/cproba.v`	Conditional probability	Sect. 2. 3
`information_theory/chap2.v`	Information theory	Sections 3. 1, 3. 2, and 3. 4
`toy_examples/conditional_entropy.v`	Example of computation	Sect. 3. 1. 2. 1
`probability/convex_choice.v` `information_theory/convex_fdist.v`	Convexity properties	Sect. 3. 3
`probability/cinde.v`	Conditional independence	Sect. 4

Systems. Cambridge University Press, 2011. 2nd edition.

[11] G. Gonthier, A. Mahboubi, and E. Tassi. A small scale reflection extension for the Coq system. Technical report, INRIA, 2008. Version 17 (Nov 2016).

[12] G. Gonthier, A. Asperti, J. Avigad, Y. Bertot, C. Cohen, F. Garillot, S. Le Roux, A. Mahboubi, R. O'Connor, S. Ould Biha, I. Pasca, L. Rideau, A. Solovyev, E. Tassi, and L. Théry. A Machine-Checked Proof of the Odd Order Theorem. In *4th International Conference on Interactive Theorem Proving (ITP 2013), Rennes, France, July 22–26, 2013*, volume 7998 of *Lecture Notes in Computer Science*, pages 163–179. Springer, 2013.

[13] O. Hasan and S. Tahar. Reasoning about conditional probabilities in a higher-order-logic theorem prover. *Journal of Applied Logic*, 9(1):23–40, 2011.

[14] J. Hölzl. *Construction and Stochastic Applications of Measure Spaces in Higher-Order Logic*. PhD thesis, Technische Universität München Institut für Informatik, 2012.

[15] B. Jacobs. Convexity, Duality and Effects. In *Theoretical Computer Science*, volume 323 of *IFIP Advances in Information and Communication Technology*, pages 1–19. Springer, 2010.

[16] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[17] X. Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115. ACM, 2009.

[18] A. Mahboubi and E. Tassi. *Mathematical Components*. Available at: `https://math-comp.github.io/mcb/` (last access: 2019/09/29), 2016. With contributions by Y. Bertot and G. Gonthier. Version of 2018/08/11.

[19] T. Mhamdi, O. Hasan, and S. Tahar. Formalization of entropy measures in HOL. In *2nd International Conference on Interactive Theorem Proving (ITP 2011), Berg en Dal, The Netherlands, August 22–25, 2011*, volume 6898 of *Lecture Notes in Computer Science*, pages 233–248. Springer, 2011.

[20] D. A. C. V. Moreira. Finite probability distributions in Coq. Master's thesis, Universidade do Minho, Escola de Engenharia, 01 2012. `http://mei.di.uminho.pt/sites/default/files/dissertacoes//eeum_di_dissertacao_pg16019.pdf` (last access: 2019/09/29).

[21] J. Pearl and A. Paz. GRAPHOIDS: A graph-based logic for reasoning about relevance relations. Technical Report R-53 850038, UCLA Computer Science Department, Dec 1985.

[22] StackExchange. What does the decomposition, weak union and contraction rule mean for conditional probability and what are their proofs? `https://math.stackexchange.com/questions/855002/what-does-the-decomposition-weak-union-and-contraction-rule-mean-for-conditiona` (last access: 2019/06/13)

[23] R. Yamaguchi, K. Kin, S. Shimoyama, M. Hagiwara, M. Yamamoto, and J. Wang. Formalization of the conditional independence using Coq/SSReflect. Technical Reports of Mathematical Sciences, Chiba University, 29:1–40, Department of Mathematics and Informatics Graduate School of Science, Chiba University, Japan, 2016. In Japanese.