Noname manuscript No.
(will be inserted by the editor)

Formalization of Shannon's Theorems

Reynald Affeldt · Manabu Hagiwara · Jonas Sénizergues

the date of receipt and acceptance should be inserted later

Abstract The most fundamental results of information theory are Shannon's theorems. These theorems express the bounds for (1) reliable data compression and (2) data transmission over a noisy channel. Their proofs are non-trivial but are rarely detailed, even in the introductory literature. This lack of formal foundations is all the more unfortunate that crucial results in computer security rely solely on information theory: this is the so-called "unconditional security". In this article, we report on the formalization of a library for information theory in the SSReflect extension of the Coq proof-assistant. In particular, we produce the first formal proofs of the source coding theorem, that introduces the entropy as the bound for lossless compression, and of the channel coding theorem, that introduces the capacity as the bound for reliable communication over a noisy channel.

1 Introduction

"Information theory answers two fundamental questions in communication theory: What is the ultimate data compression (answer: the entropy H), and what is the ultimate transmission rate of communication (answer: the channel capacity C)." This is the very first sentence of the reference book on information theory by Cover and Thomas [8]. This article is precisely about the formalization of Shannon's theorems that answer these two fundamental questions.

This article is a revised and extended version of a conference paper [1].

This work was essentially carried out when the second and third author were affiliated with Research Institute for Secure Systems, National Institute of Advanced Industrial Science and Technology, Japan.

R. Affeldt
Research Institute for Secure Systems, National Institute of Advanced Industrial Science and Technology, Japan

M. Hagiwara
Dept. of Mathematics and Informatics, Faculty of Science, Chiba University, Japan

J. Sénizergues
École Normale Supérieure de Cachan, France

The proofs of Shannon's theorems are non-trivial but are rarely detailed, let alone formalized, even in the introductory literature. Shannon's original proofs [19] in 1948 are well-known to be informal and incomplete: the proof of the channel coding theorem was not made rigorous until much later [8, Sect. 7.7], Shannon does not prove, even informally, the converse part of the channel coding theorem [22, Sect. III.A]. Though rigorous proofs are now available, the bounds that appear in Shannon's theorems (these theorems are asymptotic) are never made explicit and their existence is seldom proved carefully.

This lack of formal foundations is all the more unfortunate that several results in computer security rely crucially on information theory: this is the so-called field of "unconditional security" (one-time pad protocol, evaluation of information leakage, key distribution protocol over a noisy channel, etc.). A formalization of information theory would be a first step towards the verification of cryptographic systems based on unconditional security, and, more generally, towards the rigorous design of critical communication devices.

In this article, our first contribution is to provide a library of formal definitions and lemmas for information theory. First, we formalize finite probability, up to the weak law of large numbers, and apply this formalization to the formalization of basic information-theoretic concepts such as the entropy and typical sequences. This line of work has already been investigated by Hasan et al. [13, 16, 17] and by Coble [6], with the HOL proof-assistant. The originality of our library (besides the fact that we are working with the Coq proof-assistant [7]) lies in the formalization of advanced concepts such as jointly typical sequences, channels, codes, the so-called "method of types", etc., that are used to state and prove Shannon's theorems.

Our second and main contribution is to provide the first formal proofs of Shannon's theorems. The first Shannon's theorem is also known as the source coding theorem. This theorem introduces the entropy as the bound for lossless compression. Precisely, the direct part of this theorem shows that, given a source of information, there exist codes that compress information with negligible error rate if the rate "compressed bitstring length / original message length" is bigger than the entropy of the source of information. Conversely, any code with a rate smaller than the entropy has non-negligible error rate.

The second Shannon's theorem is also known as the channel coding theorem. It is the most famous but also the most difficult of Shannon's theorems. This theorem introduces the channel capacity as the bound for reliable communication over a noisy channel. Like the source coding theorem, the channel coding theorem comes as a direct part and a converse part. Let us consider the following explanatory scenario. Alice sends a long message to Bob over a noisy channel. Without any error-correcting code, Bob receives a message that is different from Alice's original message with probability almost 1, while its transmission rate "original message length / encoded message length" is equal to 1. On the other hand, with sufficiently repeated codes, i.e., 0 is encoded as $00\cdots0$, Bob may correctly guess the original message from the received one with failure probability almost 0, while its transmission rate is about 0. One may expect that there is a trade-off relation between the failure rate and the transmission rate. The channel coding theorem and its converse clarify this relation: (channel coding theorem) there exists a threshold C (the "capacity") such that it is possible to achieve failure rate almost 0 with an error-correcting code of any transmission rate less than C, but (converse theorem)

it is always the case that the failure rate is almost 1 with any error-correcting code of any transmission rate greater than C. Note that we are here talking about the *strong converse* of the channel coding theorem, which is the theorem that we formalized in this article. The *weak converse* of the channel coding theorem shows that failure rate cannot be 0 but does not go as far as establishing that it is almost 1. This is the simpler weak converse that one usually finds in a standard textbook such as the one by Cover and Thomas [8].

The formalization of Shannon's theorems is not a trivial matter because, in addition to the complexity of a theorem such as the channel coding theorem, the literature does not provide proofs that are organized in a way that facilitates formalization. Most importantly, it is necessary to rework the proofs so that the (asymptotic) bounds can be formalized. Indeed, information theorists often resort to claims such as "this holds for n sufficiently large", but there are in general several parameters that are working together so that one cannot choose one without checking the others. This aspect of the proofs is often ignored in the literature so that the precise formal definition of such bounds in this article can be regarded as one of our technical contributions (see Sect. 4.2, Sect. 4.3, Sect. 6.3, or Sect. 11.2 for examples). Another kind of approximation that matters when formalizing is the type of arguments. For example, in the proof of the source coding theorem, it is mathematically important to treat the source rate as a rational and not as a real. Similarly, the use of extended reals ($\pm\infty$) in the proof of the strong converse of the channel coding theorem calls for special care. Indeed, extended reals provide an intuitive way to shorten pencil-and-paper proofs but, and this is easily overlooked in an informal setting, divergence conditions need to be spelled out precisely when it comes to formalization. As a matter of fact, extended reals alone are already notoriously difficult to formalize [14, 17].

In order to ease formalization, we make several design decisions to reduce the number of concepts involved. For example, we avoid explicit use of conditional probabilities, except for the definition of discrete channels. In the proof of the strong converse of the channel coding theorem, we avoid extended reals by sorting out the situations in which divergence occurs (see Sect. 9). These design decisions do not impair readability because there are less definitions and, since proofs are more "to the point", this even contributes to a better informal understanding.

We carried out our formalization in the SSReflect extension [11] of the Coq proof-assistant [7]. Information theory involves many calculations with "big operator" notations ($\prod_i$, $\sum_i$, $\bigcup_i$, etc. notations) indexed over various kinds of sets (tuples, functions, etc.). SSReflect's library happens to provide a generic library of canonical big operators [5] together with libraries for finite sets, functions over a finite domain, etc. that can be used as index structures. The availability of these libraries will turn out to be instrumental in order to achieve reasonably-sized formal proofs for Shannon's theorems.

All the formal definitions and lemmas that appear in this article are copied verbatim from the Coq scripts (available at [2]) but are processed using the listings package of LaTeX to enhance the presentation and improve reading with colors and standard non-ASCII characters.

Outline In Sect. 2, we formalize definitions and properties about finite probability to be used in the rest of the article. In Sect. 3, we introduce the concepts of entropy and typical sequence. In Sect. 4, we state the source coding theorem and detail

in particular the proof of its direct part. In Sect. 5, we formalize the concept of noisy channel and illustrate related information-theoretic definitions thoroughly using the example of the binary symmetric channel. In Sect. 6, we state and prove the direct part of the channel coding theorem. Sect. 7 explains the proof approach for the (strong) converse of the channel coding theorem. In Sect. 8, we introduce the basic definitions of the method of types. We start proving basic results about types in Sect. 9 where we define in particular the concept of information divergence. Sect. 10 is a concrete application of the method of types, that we use to upper-bound the success rate of decoding. The converse of the channel coding theorem is proved in Sect. 11. Sect. 12 provides a last result that is crucial to complete the proof of the converse of the channel coding theorem. It is isolated because it is technical and its proof is of broader interest than the sole channel coding theorem. Sect. 13 provides a quantitative overview of the formalization. Sect. 14 is dedicated to related work.

2 The Basics: Finite Probability

We introduce basic definitions about probability (to explain the notations to be used in this article) and formalize the weak law of large numbers. We do not claim that this formalization is a major contribution in itself because there exist more general formalizations of probability theory (in particular in the HOL proof-assistant [13,16,17]) but providing a new formalization using SSReflect will allow us to take advantage of its library to prove Shannon's theorems.

2.1 Probability Distributions

A distribution over a finite set `A` (in practice, a type of type `finType` in SSReflect) will be formalized as a real-valued probability mass function `pmf` with positive outputs that sum to 1. In this setting, the probability space is the powerset of `A`. An event is a subset of the probability space (it can therefore be encoded as a set of type `{set A}` in SSReflect). Let us first define real-valued functions with positive outputs:

```
0 Record pos_fun (T : Type) := mkPosFun {
1   pos_f :> T → R ;
2   Rle0f : ∀ a, 0 ≤ pos_f a }.
```

`R` is the type of reals in the Coq standard library. `pos_f` (line 1) is a function from some type `T` to `R`. `Rle0f` (line 2) is the proof that all the outputs are positive. An object `f` of type `pos_fun T` is a `Record` but, thanks to the coercion `:>` at line 1, we can write "`f a`" as a standard function application. Below, we denote the type of positive functions over `T` by `T →R+`. A distribution can now be formalized as a positive function whose outputs sum to 1:

```
0 Record dist := mkDist {
1   pmf :> A → R+ ;
2   pmf1 : Σ_(a in A) pmf a = 1 }.
```

At line 1, `pmf` (for "probability mass function") is a positive function. At line 2, `pmf1` is the proof that all the outputs sum to 1 (the big sum operator is an instance of the

canonical big operators of SSReflect [5]). We use again the coercion mechanism (:> at line 1) so that we can write "P a" as a function application to represent the probability associated with a despite the fact that the object P of type dist A is actually a Record.

We will be led to define several kinds of distributions in the course of this article. Here is the first example. Given distributions P1 over A and P2 over B, the *product distribution* P1 × P2 over A * B is defined as d below (inside a module ProdDist):

```
Definition f (ab : A * B) := P1 ab.1 * P2 ab.2. (* prob. mass fun. *)
Lemma f0 (ab : A * B) : 0 ≤ f ab.
Lemma f1 : Σ_(ab | ab ∈ {: A * B}) f ab = 1.
Definition d : dist [finType of A * B] := makeDist f0 f1.
```

The .1 (resp. .2) notation is for the first (resp. second) pair projection. The notation [finType of ...] is just a type cast so that [finType of A * B] can simply by thought as the cartesian product A * B.

Given a distribution P over A, the probability of an event E (encoded as a set of elements of type A) is defined as follows:

```
Definition Pr P (E : {set A}) := Σ_(a in E) P a.
```

2.2 Random Variables

We formalize a random variable as a Record containing a distribution (rv_dist below) coupled with a real-valued function (rv_fun below):

```
Record rvar A := mkRvar {
  rv_dist : dist A ;
  rv_fun :> A → R }.
```

This definition is sufficient for our purpose because A will always by finite in this article. Again, thanks to the coercion, given a random variable X and a in A, one can write "X a" as in standard mathematical writing despite the fact that X is actually a Record. Hereafter, we denote the distribution underlying the random variable X by p_X.

We formalize the probability that a random variable X evaluates to some real r as follows:

```
Definition pr (X : rvar A) r := Pr p_X [set x | X x = r].
```

[set x | P x] is an SSReflect notation for the set of elements that satisfy the boolean predicate P. Hereafter, we will use the traditional notation Pr[X = r] for pr X r.

Given a random variable X over A, and writing img X for its image, we define the expected value as follows:

```
Definition Ex X := Σ_(r ← img X) r * Pr[X = r].
```

In the following, $\mathcal{E}$ X denotes the expected value of the random variable X.

Let us now define the sum of random variables. Below, n.-tuple A is the SSReflect type for n-tuples over A, written A^n using standard mathematical notations.

Let us assume the distribution P1 over A, the distribution P2 over n.-tuple A, and the distribution P over n+1.-tuple A. P is a *joint distribution* for P1 and P2 when its marginal distributions satisfy the following predicate:

```
Definition joint P1 P2 P :=
  (∀ x, P1 x = Σ_(t in {:n+1.-tuple A} | thead t = x) P t) ∧
  (∀ x, P2 x = Σ_(t in {:n+1.-tuple A} | tbehead t = x) P t).
```

In other words, `joint P1 P2 P` is a relation that defines the distribution `P1` (resp. `P2`) from the distribution `P` by taking into account only the first element (resp. all the elements but the first) of the tuples from the sample space (`thead` returns the first element of a tuple; `tbehead` returns all the elements but the first).

The random variable `X` is the sum of `X1` and `X2` when the distribution of `X` is the joint distribution of the distributions of `X1` and `X2` and the output of `X` is the sum of the outputs of `X1` and `X2`:

```
Definition sum := joint p_X1 p_X2 p_X ∧
  X =_1 fun x ⇒ X1 (thead x) + X2 (tbehead x).
```

$=_1$ is the extensional equality for functions.

The random variables `X` over `A` and `Y` over `n.-tuple A` are *independent* for a distribution `P` (over `n+1.-tuple A`) when the following predicate holds:

```
Definition inde_rv := ∀ x y,
  Pr P [set xy | (X (thead xy) = x) ∧ (Y (tbehead xy) = y)] =
  Pr[X = x] * Pr[Y = y].
```

Similarly to [13], we define the sum of several random variables by generalizing the definition `sum` to an inductive predicate, namely `sum_n`. More precisely, let `Xs` be a tuple of `n` random variables over `A` and `X` be a random variable over `n.-tuple A`: `sum_n Xs X` holds when the random variable `X` is the sum of the random variables in `Xs`. We also specialize this definition to the sum of independent random variables by using the predicate `inde_rv` above. Equipped with above definitions, we derive the standard properties of the expected value, such as its linearity, but also properties of the variance. See [2] for details.

2.3 The Weak Law of Large Numbers

The weak law of large numbers is the first fundamental theorem of probability. Intuitively, it says that the average of the results obtained by repeating an experiment a large number of times is close to the expected value. Formally, let `Xs` be a tuple of `n+1` *identically distributed* random variables, i.e., random variables with the same distribution `P`. Let us assume that these random variables are independent and let us write `X` for their sum, μ for their common expected value, and σ^2 for their common variance. The weak law of large numbers says that the outcome of the average random variable `X / (n+1)` gets closer to μ:

```
Lemma wlln ε : 0 < ε →
  Pr p_X [set t | Rabs ((X / (n+1)) t - μ) ≥ ε] ≤ σ^2 / ((n+1) * ε ^ 2).
```

`Rabs` is the absolute value in the Coq standard library. See [2] for the proof of this lemma using the Chebyshev inequality.

3 Entropy and Typical Sequences

We formalize the central concept of a typical sequence. Intuitively, a typical sequence is a tuple of n symbols that is expected to be observed when n is large. For

example, a tuple produced by a binary source that emits 0's with probability 2/3 is typical when it contains approximately two thirds of 0's. The precise definition of typical sequences requires the definition of the entropy and their properties rely on a technical result known as the Asymptotic Equipartition Property. (Mhamdi et al. [17] provide an alternative HOL version of most of the definitions and properties in this section.)

3.1 Entropy and Asymptotic Equipartition Property

We define the entropy of a random variable with distribution `P` over `A` as follows (where log is the binary logarithm, derived from the Coq standard library):

```
Definition entropy P := - Σ_(a in A) P a * log (P a).
```

In the following, $\mathcal{H}$`P` denotes the entropy of `P`. Let $-\log$ `P` be the random variable defined as follows:

```
Definition mlog_rv P := mkRvar P (fun x ⇒ - log (P x)).
```

We observe that $\mathcal{H}$`P` is actually equal to the expected value of the random variable $-\log$ `P`:

```
Lemma entropy_Ex P : H P = E (- log  P).
```

The Asymptotic Equipartition Property (AEP) is a property about the outcomes of n random variables $-\log$ `P` that are independent and identically distributed (with distribution `P`). The probability in the AEP is taken over a *tuple distribution*. Given a distribution `P` over `A`, the tuple distribution `P^n` over `n.-tuple A` is defined as `d` below (inside a module `TupleDist`):

```
Definition f (t : n.-tuple A) := Π_(i < n) P t_i. (* prob. mass fun. *)
Lemma f0 (t : n.-tuple A) : 0 ≤ f t.
Lemma f1 : Σ_(t | t ∈ {:n.-tuple A}) f t = 1.
Definition d : dist [finType of n.-tuple A] := makeDist f0 f1.
```

The big product operator is another instance of SSReflect's canonical big operators. `t_i` is the `i`th element of the tuple `t`.

Informally, the AEP states that, in terms of probability, the outcome of the random variable $-\log$ `(P^(n+1)) /(n+1)` is "close to" the entropy $\mathcal{H}$`P`. Here, "close to" means that, given an $\varepsilon >$`0` and a tuple `t` of length `n`, the probability that the outcome $(-\log$ `(P^(n+1)) /(n+1))` `t` and $\mathcal{H}$`P` differ by more than ε is less than ε, when `n+1` is greater than the bound `aep_bound P` ε defines as follows:

```
Definition aep_σ² P := Σ_(a in A) P a * (log (P a))^2 - (H P)^2.
Definition aep_bound P ε := aep_σ² P / ε^3.
```

Using above definitions, the AEP can now be stated formally. Its proof is an application of the weak law of large numbers (Sect. 2.3):

```
Lemma aep : aep_bound P ε ≤ n+1 →
  Pr (P^(n+1)) [set t | (0 < P^(n+1) t) ∧
                  (Rabs ((- log  (P^(n+1)) / (n+1)) t - H P) ≥ ε) ] ≤ ε.
```

3.2 Typical Sequences: Definition and Properties

Given a distribution P over A and some ε, a typical sequence is an n-tuple t with probability "close to" $2^{-n\mathcal{H}P}$, as captured by the following predicate:

```
Definition typ_seq (t : n.-tuple A) :=
  exp (- n * (ℋ P + ε)) ≤ P^n t ≤ exp (- n * (ℋ P - ε)).
```

We denote the set of typical sequences by $\mathcal{TS}$. Using the AEP, we prove that the probability of the event $\mathcal{TS}$ for large n is close to 1, corresponding to the intuition that a typical sequence is expected to be observed in the long run:

```
Lemma Pr_TS_1 : aep_bound P ε ≤ n+1 → Pr (P^(n+1)) (𝒯𝒮 P n+1 ε) ≥ 1 - ε.
```

Recall that aep_bound has been defined in the previous section (Sect. 3.1).

The cardinal of $\mathcal{TS}$ is nearly $2^{n\mathcal{H}P}$. Precisely, it is upper-bounded by $2^{n(\mathcal{H}P+\varepsilon)}$, and lower-bounded by $(1-\varepsilon)2^{n(\mathcal{H}P-\varepsilon)}$ for n big enough:

```
Lemma TS_sup : | 𝒯𝒮 P n ε | ≤ exp (n * (ℋ P + ε)).
Lemma TS_inf : aep_bound P ε ≤ n+1 →
  (1 - ε) * exp ((n+1) * (ℋ P - ε)) ≤ | 𝒯𝒮 P n+1 ε |.
```

4 The Source Coding Theorem

The source coding theorem (a.k.a. the noiseless coding theorem) is a theorem for data compression. The basic idea is to replace frequent words with alphabet sequences and other words with a special symbol. Let us illustrate this with an example. The combination of two Roman alphabet letters consists of 676 ($= 26^2$) words. Since $2^9 < 676 < 2^{10}$, 10 bits are required to represent all the words. However, by focusing on often-used English words ("as", "in", "of", etc.), we can encode them with less than 9 bits. Since this method does not encode rarely-used words (such as "pz"), decoding errors can happen. Given an information source known as a discrete memoryless source (DMS) that emits all symbols with the same distribution P, the source coding theorem gives a theoretical lower-bound (namely, the entropy $\mathcal{H}$P) for compression rates with negligible error rate.

4.1 Definition of a Source Code

Given a set A of symbols, a k,n-source code is a pair of an encoder and a decoder. The encoder maps a k-tuple of symbols to an n-tuple of bits and the decoder performs the corresponding decoding operation:

```
Definition encT := k.-tuple A → n.-tuple bool.
Definition decT := n.-tuple bool → k.-tuple A.
Record scode := mkScode { enc : encT ; dec : decT }.
```

The rate of a k,n-source code sc is defined as the ratio of bits per symbol:

```
Definition SrcRate (sc : scode) := n / k.
```

Given a DMS with distribution P over A, the error rate of a source code sc (notation: $\bar{e}_{src}$(P, sc)) is defined as the probability of failure for the decoding of encoded sequences:

```
Definition SrcErrRate P sc := Pr (P^k) [set t | dec sc (enc sc t) ≠ t].
```

4.2 Source Coding Theorem—Direct Part

Given a source of symbols from the alphabet `A` with distribution `P`, there exist source codes of rate `r` $\in \mathcal{Q}^+$ (the positive rationals) larger than the entropy $\mathcal{H}$`P` such that the error rate can be made arbitrarily small:

```
Theorem source_coding_direct : ∀ ε, 0 < ε < 1 →
  ∀ r : Q+, H P < r →
    ∃ k n (sc : scode A k n), SrcRate sc = r ∧ ē_src(P , sc) ≤ ε.
```

Source Coding using the Typical Set The crux of the proof is to instantiate with an adequate source code. For this purpose, we first define a generic encoder and a generic decoder, both parameterized by a set `S` of `k+1`-tuples. In the proof of the source coding theorem, we will actually take the set `S` to be the set $\mathcal{TS}$ of "long enough" typical sequences, but we need to explain the encoding and decoding strategy before giving a precise meaning to "long enough".

The encoder function `f` encodes the ith element of `S` as the binary encoding of $i+1$, and elements not in `S` as a string of 0's:

```
Definition f : encT A k+1 n := fun x ⇒
  if x ∈ S then
    let i := index x (enum S) in Tuple (size_nat2bin i+1 n)
  else
    [tuple of nseq n false].
```

`enum S` is the list of all the elements of `S`. `index` returns the index of an element in a list. `Tuple (size_nat2bin i n)` is a tuple of size `n` that contains the binary encoding of `i` $< 2^{\mathtt{n}}$. Last, `nseq n false` is a list of n `false` booleans, where `false` represents the bit 0. `[tuple of ...]` is just a type cast and can be ignored.

The definition of the decoder function requires a default element `def` $\in$ `S`. Given a bitstring `x`, the decoder function ϕ interprets `x` as a natural number i and returns the $(i-1)^{\text{th}}$ element of `S` if i is smaller than the cardinal of `S`, or the default value `def` otherwise:

```
Definition φ : decT A k+1 n := fun x ⇒
  let i := tuple2N x in
  if i is 0 then def else
    if i-1 < |S| then nth def (enum S) i-1 else def.
```

`tuple2N` interprets bitstrings as natural integers and `nth` picks up the nth element of a list.

By construction, when `| S |` $<$ `2 ^ n`, `f` and ϕ perform lossless coding only for elements in `S`:

```
Lemma φ_f i : φ (f i) = i ↔ i ∈ S.
```

As we have already said above, in the proof of the source coding theorem, we actually take `S` to be the set $\mathcal{TS}$ of "long enough" typical sequences. Here, "long enough" means long enough to guarantee the existence of a default element `def`. This is achieved by taking `k` bigger than some bound that we now make precise.

Formalization of the Bound Above, we explained how to construct the required source code. Technically, in the formal proof, it is also important to correctly instantiate `n` and `k` (given the source rate `r`, the real ε and the distribution `P`), such that `k` is "big enough" for the lemma ϕ_`f` to hold.

Let us define the following quantities:

```
Definition λ := min(r - ℋ P, ε).
Definition δ := max(aep_bound P (λ / 2), 2 / λ).
```

`k` must satisfy $\delta \leq$ `k` and `k * r` must be a natural. Such a `k` can be constructed using the following lemma:

```
Lemma SrcDirectBound d D : {k | D ≤ (k+1) * (d+1)}.
```

`{k | P k}` is a standard Coq notation for existential quantification and can be read as $\exists$`k`.`P k` as far as we are concerned.

Let us assume that the rate is `r = num / (den+1)`. We denote by `k'` the natural constructed via the above lemma by taking `d` to be the denominator `den` and `D` to be δ. Then it is sufficient to take `n` equal to `(k'+1) * num` and `k` equal to `(k'+1) * (den+1)`.

Finally, we can instantiate the generic encoder and decoder functions by taking the set `S` to be the set $\mathcal{TS}$ `P k (λ / 2)`.

At this point, we have thoroughly explained how to instantiate the source code required by the source coding theorem. The proof is completed by appealing to the properties of typical sequences, in particular, lemmas `Pr_TS_1` and `TS_sup` from Sect. 3.2. The successive steps of the proof can be found in [2].

4.3 Source Coding Theorem—Converse Part

The converse of the Shannon's source coding theorem shows that any source code whose rate is smaller than the entropy of a source with distribution `P` over `A` has non-negligible error rate:

```
Theorem source_coding_converse : ∀ ε, 0 < ε < 1 →
  ∀ r : 𝒬⁺, 0 < r < ℋ P →
    ∀ n k (sc : scode A k+1 n),
      SrcRate sc = r →
      SrcConverseBound P (num r) (den r) n ε ≤ k+1 →
      ē_src(P , sc) ≥ ε.
```

`num r` (resp. `den r`) is the numerator (resp. denominator) of `r`. The bound given by `SrcConverseBound` gives a precise meaning to the claim that would otherwise be informally summarized as "for `k` big enough":

```
Definition λ := min((1 - ε) / 2, (ℋ P - r) / 2).
Definition δ := min((ℋ P - r) / 2, λ / 2).
Definition SrcConverseBound := max(max(
  aep_bound P δ, - ((log δ) / (ℋ P - r - δ))), n / r).
```

The proof of the converse part of the source coding theorem is a bit simpler than the direct part because no source code needs to be constructed. See [2] for the detail of the proof steps.

5 Formalization of Channels

As a first step towards the formalization of the channel coding theorem, the goal of this section is to introduce the formalization of channels (Sect. 5.1), the formalization of the channel capacity (Sect. 5.2), and the formalization of jointly typical sequences (Sect. 5.4).

5.1 Discrete Memoryless Channel

A *discrete channel* with input alphabet `A` and output alphabet `B` is a (probability transition) matrix that expresses the probability of observing an output symbol given some input symbol:

$$\begin{bmatrix} W(b_1|a_1) & W(b_2|a_1) & \cdots & W(b_{|B|}|a_1) \\ W(b_1|a_2) & W(b_2|a_2) & \cdots & W(b_{|B|}|a_2) \\ \vdots & \vdots & \ddots & \vdots \\ W(b_1|a_{|A|}) & W(b_2|a_{|A|}) & \cdots & W(b_{|B|}|a_{|A|}) \end{bmatrix}$$

One way to view such a matrix is as a function that associates to each input $a_i \in A$ a distribution of the corresponding outputs as the matrix row $W(b_1|a_i)$, $W(b_2|a_i)$, ..., $W(b_{|B|}|a_i)$. Therefore, we formalize channels as functions returning distributions and denote the type `A` $\to$ `dist B` by $\mathcal{CH}_1$ `(A, B)`.

In the second part of this article (starting at Sect. 8), it will sometimes be convenient to simplify the presentation by considering channels whose input alphabet is not empty (other channels are not interesting anyway):

```
Record chan_star := mkChan {
  c :> CH_1 (A, B) ;
  input_not_0 : 0 < |A| }.
```

We will denote `chan_star A B` by $\mathcal{CH}_1{}^*$ `(A, B)`.

The *nth extension of a discrete channel* is the generalization of a discrete channel to the communication of n symbols:

```
Definition channel_ext n := n.-tuple A → dist [finType of n.-tuple B].
```

Hereafter, we denote `channel_ext n` by $\mathcal{CH}_{\mathtt{n}}$.

A *discrete memoryless channel* (DMC) models channels whose inputs do not depend on past outputs. It is the special case of the nth extension of a discrete channel. Using a channel `W`, we define a DMC as `c` below (in a module `DMC`):

```
Definition f (ta : n.-tuple A) (tb : n.-tuple B) := Π_(i < n) W ta_i tb_i.
Lemma f0 (ta : n.-tuple A) (tb : n.-tuple B) : 0 ≤ f ta tb.
Lemma f1 (ta : n.-tuple A) : Σ_( tb | tb ∈ {: n.-tuple B}) f ta tb = 1.
Definition c : CH_n := fun ta ⇒ makeDist (f0 ta) (f1 ta).
```

Hereafter, we denote `DMC.c W n` by `W^n`.

5.2 Mutual Information and Channel Capacity

Given a discrete channel `W` with input alphabet `A` and output alphabet `B`, and an input distribution `P`, there are two important distributions: the output distribution and the joint distribution. The *output distribution* (notation: $\mathcal{O}$`(P , W)`) is the distribution of the outputs defined as `d` below (in a module `OutDist`):

```
Definition f (b : B) := Σ_(a in A) W a b * P a. (* prob. mass fun. *)
Lemma f0 (b : B) : 0 ≤ f b.
Lemma f1 : Σ_(b in B) f b = 1.
Definition d : dist B := makeDist f0 f1.
```

The *joint distribution* (notation: `𝒥(P , W)`) is the joint distribution of the inputs and the outputs defined as `d` below (in a module `JointDist`):

```
Definition f (ab : A * B) := W ab.1 ab.2 * P ab.1. (* prob. mass fun. *)
Lemma f0 (ab : A * B) : 0 ≤ f ab.
Lemma f1 : Σ_(ab | ab ∈ {: A * B}) (W ab.1) ab.2 * P ab.1 = 1.
Definition d : dist [finType of A * B] := makeDist f0 f1.
```

The *output entropy* (resp. *joint entropy*) is the entropy of the output distribution (resp. joint distribution), hereafter denoted by `ℋ(P ∘ W)` (resp. `ℋ(P , W)`). The *conditional entropy* `ℋ(W | P)` (the entropy of the output knowing the input) is defined using the joint entropy:

```
Definition cond_entropy P (W : 𝒞ℋ_1(A, B)) := ℋ(P , W) - ℋ P.
```

The *mutual information* (notation: `ℐ(P ; W)`) is a measure of the amount of information that the output distribution contains about the input distribution:

```
Definition mut_info P (W : 𝒞ℋ_1(A, B)) := ℋ P + ℋ(P ∘ W) - ℋ(P , W).
```

Finally, the *information channel capacity* is defined as the least upper bound of the mutual information taken over all possible input distributions:

```
Definition ubound {S : Type} (f : S → R) (ub : R) := ∀ a, f a ≤ ub.
Definition lubound {S : Type} (f : S → R) (lub : R) :=
  ubound f lub ∧ ∀ ub, ubound f ub → lub ≤ ub.
Definition capacity (W : 𝒞ℋ_1(A, B)) cap := lubound (fun P ⇒ ℐ(P ; W)) cap.
```

It may not be immediate why the maximum mutual information is called capacity. The goal of the channel coding theorem is to ensure that we can distinguish between two outputs (actually sets of outputs because of potential noise), so as to be able to deduce the corresponding inputs without ambiguity. For each input (of n symbols), there are approximately $2^{n\mathcal{H}(W|P)}$ typical outputs because $\mathcal{H}(W|P)$ is the entropy of the output knowing the input. On the other hand, the total number of typical outputs is approximately $2^{n\mathcal{H}(P\circ W)}$. Since this set has to be divided into sets of size $2^{n\mathcal{H}(W|P)}$, the total number of disjoint sets is less than or equal to $2^{n(\mathcal{H}(P\circ W)-\mathcal{H}(W|P))} = 2^{n\mathcal{I}(P;W)}$.

5.3 Example: The Binary Symmetric Channel

We illustrate above definitions with a simple model of channel with errors: the `p`-binary symmetric channel. In such a channel, the input and output symbols are taken from the same alphabet `A` with only two symbols (hypothesis `card_A` below). Upon transmission, the input is flipped with probability `p` (we assume that we are working under the hypothesis `p_01 : 0 ≤ p ≤ 1`). We define a binary symmetric channel as `c` below (in a module `BSC`):

```
Hypothesis card_A : |A| = 2.
Hypothesis p_01 : 0 ≤ p ≤ 1.
Definition f (a : A) := fun a' ⇒ if a = a' then 1 - p else p.
Lemma f0 (a a' : A) : 0 ≤ f a a'.
Lemma f1 (a : A) : Σ_(a' | a' ∈ A) f a a' = 1.
Definition c : 𝒞ℋ_1(A, A) := fun a ⇒ makeDist (f0 a) (f1 a).
```

For convenience, we introduce the *binary entropy function*:

```
Definition ℋ_2 p := - p * log p - (1 - p) * log (1 - p).
```

For any input distribution P, we prove that the mutual information can actually be expressed by only the entropy of the output distribution and the binary entropy function:

```
Lemma IPW : ℐ(P ; BSC.c card_A p_01) = ℋ(P ∘ BSC.c card_A p_01) - ℋ_2 p.
```

The maximum of the binary entropy function on the interval $]0,1[$ is 1, fact that we proved formally in Coq by appealing to the standard library for reals[1]:

```
Lemma ℋ_2_max : ∀ p, 0 < p < 1 → ℋ_2 p ≤ 1.
```

This fact gives an upper-bound for the entropy of the output distribution:

```
Lemma H_out_max : ℋ(P ∘ BSC.c card_A p_01) ≤ 1.
```

The latter bound is actually reached for the uniform input distribution. Let us first define uniform distributions for any non-empty fintype B as d below (in a module Uniform):

```
Variable B : finType.
Variable n : nat.
Hypothesis Bnot0 : |B| = n+1.
Definition f (b : B) := 1 / |B|. (* prob. mass fun. *)
Lemma f0 b : 0 ≤ f b.
Lemma f1 : Σ_(b | b ∈ B) f b = 1.
Definition d : dist B := makeDist f0 f1.
```

The fact the upper-bound 1 is reached for the uniform input distribution is captured by the following lemma:

```
Lemma H_out_binary_uniform :
  ℋ(Uniform.d card_A ∘ BSC.c card_A p_01) = 1.
```

Above facts imply that the capacity of the p-binary symmetric channel can be expressed by a simple closed formula:

```
Theorem BSC_capacity : capacity (BSC.c card_A p_01) (1 - ℋ_2 p).
```

5.4 Jointly Typical Sequences

Let us consider a channel W with input alphabet A, output alphabet B, and input distribution P, and some ε. A *jointly typical sequence* is a pair of two sequences such that: (1) the first sequence is typical for P, (2) the second sequence is typical for the output distribution $\mathcal{O}$(P , W), and (3) the pair is typical for the joint distribution $\mathcal{J}$(P , W):

```
Definition jtyp_seq (t : n.-tuple (A * B)) :=
  typ_seq P ε [tuple of unzip1 t] ∧
  typ_seq (𝒪(P , W)) ε [tuple of unzip2 t] ∧
  typ_seq (𝒥(P , W)) ε t.
```

We denote the set of jointly typical sequences by $\mathcal{JTS}$. The number of jointly typical sequences is upper-bounded by $2^{n(\mathcal{H}(P,W)+\varepsilon)}$:

```
Lemma JTS_sup : | 𝒥𝒯𝒮 P W n ε| ≤ exp (n * (ℋ(P , W) + ε)).
```

[1] Modulo a slight extension of the corollary of the mean value theorem to handle derivability of partial functions.

Now follow two lemmas that will be key to prove the channel coding theorem.

With high probability (probability taken over the tuple distribution of the joint distribution), the sent input and the received output are jointly typical. In other words, when they are very long, the jointly typical sequences coincide with the typical sequences of the joint distribution (Fig. 1):

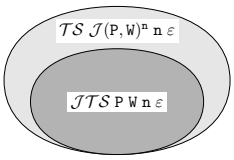

Fig. 1 Lemma JTS_1

```
Lemma JTS_1 : JTS_1_bound P W ε ≤ n →
  Pr (𝒥(P , W)^n) (𝒥𝒯𝒮 P W n ε) ≥ 1 - ε.
```

The bound JTS_1_bound is defined as follows:

```
Definition JTS_1_bound P W ε :=
  maxn (up (aep_bound P (ε / 3)))
 (maxn (up (aep_bound (𝒪(P , W)) (ε / 3)))
       (up (aep_bound (𝒥(P , W)) (ε / 3)))).
```

(up r is a function from the Coq standard library that returns the ceiling of r.) This bound will later appear again in the proof of the channel coding theorem (Sect. 6.3).

In contrast, the probability of the same event (joint typicality) taken over the product distribution of the inputs and the outputs considered independently tends to 0 as n gets large (see also Fig. 2):

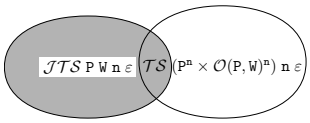

Fig. 2 Lemma non_typical_sequences

```
Lemma non_typical_sequences :
  Pr ((P^n) × ((𝒪(P , W))^n))
    [set x | x ∈ 𝒥𝒯𝒮 P W n ε] ≤
  exp (- n * (ℐ( P ; W) - 3 * ε)).
```

6 The Channel Coding Theorem

6.1 Formalization of a Channel Code

The purpose of a code is to transform the input of a channel (for practical use, by adding some form of redundancy) so that the transmitted information can be recovered correctly from the output despite of potential noise. Concretely, given input alphabet A and output alphabet B, a (channel) code is (1) a set M of messages, (2) an encoding function that turns a message into a codeword of n input symbols, and (3) a decoding function that turns n output symbols back into a message (or possibly fails):

```
Definition encT := {ffun M → n.-tuple A}.
Definition decT := {ffun n.-tuple B → option M}.
Record code := mkCode { enc : encT ; dec : decT }.
```

{ffun T →...} is the type of functions over a finite domain T.

The *rate* of a code is defined as follows:

```
Definition CodeRate (c : code) := log |M| / n.
```

For convenience, we introduce the following type to characterize (channel) code rates:

```
Record CodeRateType := mkCodeRateType {
  rate :> R ;
  _ : ∃ n d, 0 < n ∧ 0 < d ∧ rate = log (n) / d }.
```

So, a code rate is a pair of a real `rate` and a proof that `rate` has a specific form.

We now define the error rate. Given a channel `W` and a tuple of inputs `ta`, we denote the distribution of outputs knowing that `ta` was sent by `W^n (| ta)`. Using this distribution, we first define the probability of decoding error for the code `c` knowing that the message `m` was sent (notation: `e(W, c) m`):

```
Definition ErrRateCond (W : 𝒞ℋ₁(A, B)) c m :=
  Pr (W^n (| enc c m)) [set tb | dec c tb ≠ Some m].
```

Finally, we define the error rate as the average probability of error for a code `c` over channel `W` (notation: $\bar{e}_{cha}$`(W, c)`):

```
Definition CodeErrRate (W : 𝒞ℋ₁(A, B)) c :=
  1 / |M| * Σ_(m in M) e(W, c) m.
```

6.2 Channel Coding Theorem—Statement of the Direct Part

The (noisy-)channel coding theorem is a theorem for reliable information transmission over a noisy channel. The basic idea is to represent the original message by a longer message. Let us illustrate this with an example. Assume the original message is either 0 or 1 and is sent over a p-binary symmetric channel (see Sect. 5.3). The receiver obtains the wrong message with probability p. Let us now consider that the original message is 0 and encode 0 into 000 before transmission (in other words, we use a repetition encoding with code rate 1/3). The receiver obtains a message from the set $\{000, 001, 010, 100\}$ with probability $(1-p)^3 + 3p(1-p)^2$ and it guesses the original message 0 by majority vote. The error probability $1 - ((1-p)^3 + 3p(1-p)^2)$ is smaller than p.

One may guess that the smaller the code rate is, the smaller the error probability becomes. Given a discrete channel `W` (with input alphabet `A`, output alphabet `B`, and capacity `cap`—hypothesis `capacity W cap`), the channel coding theorem guarantees the existence of an encoding function and a decoding function such that the code rate is not small (yet smaller than the capacity `cap`) but is with negligible error rate:

```
Theorem channel_coding (r : CodeRateType) : r < cap →
  ∀ ε, 0 < ε →
    ∃ n M (c : code A B M n), CodeRate c = r ∧ ē_cha(W, c) < ε.
```

6.3 Channel Coding Theorem—Proof of the Direct Part

We formalize the classic proof by "random coding". Before delving into the details, let us give an overview of the proof. To prove the existence of a code suitable for the channel coding theorem, we first fix the decoding function (function `jtdec` below). We use jointly typical sequences to define this function. Then, we

select a corresponding encoding function by checking all the possible ones. Selection operates using a criterion about the average error rate of all the possible encoding functions, weighted according to a well-chosen distribution (lemma `good_code_sufficient_condition` below). Last, we show that this average error rate can be bounded using the properties of jointly typical sequences (lemmas `JTS_1` and `non_typical_sequences` from Sect. 5.4).

Decoding by Joint Typicality We first fix the decoding function `jtdec`. Given the channel output `tb`, `jtdec` looks for a codeword `m` such that the channel input `f m` is jointly typical with `tb`. If a unique such codeword is found, it is declared to be the sent codeword:

```
Definition jtdec P W ε (f : encT A M n) : decT B M n :=
  [ffun tb ⇒ [pick m |
    ((f m, tb) ∈ 𝒥𝒯𝒮 P W n ε) ∧
    [∀ m', (m' ≠ m) ⇒ ((f m', tb) ∉ 𝒥𝒯𝒮 P W n ε)]]].
```

`[ffun x ⇒...]` is a SSReflect notation to define functions over finite domains. `[pick m | P m]` is a SSReflect construct that picks up an element `m` satisfying the predicate `P`.

Criterion for Encoder Selection We are looking for a channel code such that the error rate can be made arbitrarily small. The following lemma provides a sufficient condition for the existence of such a code:

```
Lemma good_code_sufficient_condition P W ε
  (ϕ : encT A M n → decT B M n) :
  Σ_(f : encT A M n) (Wght.d P f * ē_cha(W , mkCode f (ϕ f))) < ε →
  ∃ f, ē_cha(W , mkCode f (ϕ f)) < ε.
```

In this lemma, `Wght.d P` is the distribution of encoding functions defined as follows (in a module `Wght`):

```
Definition pmf := fun f : encT A M n ⇒ Π_(m in M) P^n (f m).
Lemma pmf0 (f : {ffun M → n.-tuple A}) : 0 ≤ pmf f.
Lemma pmf1 : Σ_(f | f ∈ {ffun M → n.-tuple A}) pmf f = 1.
Definition d : dist [finType of encT A M n] := makeDist pmf0 pmf1.
```

The Main Lemma Our theorem can be derived from the following technical lemma by just proving the existence of appropriate ε_0 and `n`. This lemma establishes that there exists a set of messages `M` such that decoding by joint typicality meets the above criterion for encoder selection:

```
Lemma random_coding_good_code ε : 0 ≤ ε →
  ∀ (r : CodeRateType),
    ∀ ε0, ε0_condition r ε ε0 →
    ∀ n, n_condition r ε0 n →
  ∃ M : finType, 0 < |M| ∧ |M| = Int_part (exp (n * r)) ∧
  let Jtdec := jtdec P W ε0 in
  Σ_(f : encT A M n) (Wght.d P f * ē_cha(W , mkCode f (Jtdec f))) < ε.
```

(`Int_part x` is a function from the Coq standard library that returns the integer part of `x`.) In this lemma, the fact that the rate `r` is bounded by the mutual information appears in the condition ε_0`_condition`:

```
Definition ε0_condition r ε ε0 :=
  0 < ε0  ∧  ε0 < ε / 2  ∧  ε0 < (𝐼(P ; W) - r) / 4.
```

The condition n_condition corresponds to the formalization of the restriction "for n big enough" (we saw the bound JTS_1_bound in Sect. 5.4):

```
Definition n_condition r ε0 n :=
  0 < n  ∧  - log ε0 / ε0 < n  ∧
  frac_part (exp (n * r)) = 0  ∧  JTS_1_bound P W ε0 ≤ n.
```

(frac_part x is a function from the Coq standard library that returns the fractional part of x.)

Proof of the Main Lemma The first thing to observe is that by construction the error rate averaged over all possible encoders does not depend on which message m was sent:

```
Lemma error_rate_symmetry (P : dist A) (W : 𝒞ℋ1(A, B)) ε :
  0 ≤ ε → let Jtdec := jtdec P W ε in
    ∀ m m',
      Σ_(f : encT A M n) (Wght.d P f * e(W, mkCode f (Jtdec f)) m) =
      Σ_(f : encT A M n) (Wght.d P f * e(W, mkCode f (Jtdec f)) m').
```

Therefore, the left-handside of the conclusion of the main lemma (line 6 of the statement of the lemma random_coding_good_code above) can be rewritten by assuming that the message 0 was sent:

```
Σ_(f : encT A M n) Wght.d P f * Pr (W^n (| f 0)) (not_preimg (Jtdec f) 0)
```

where not_preimg (Jtdec f) 0 is the set of outputs that do not decode to 0.

Let us write $\mathcal{B}$ f m for the set of outputs tb such that (f m, tb) ∈ $\mathcal{JTS}$ P W n ε. Assuming that 0 was sent, a decoding error occurs when (1) the input and the output are not jointly typical, or (2) when a wrong input is jointly typical with the output. This can be expressed formally by the following set equality:

```
not_preimg (JTdec f) 0 =i  (~: ℬ f 0) ∪ ⋃_(m : M | m ≠ 0) ℬ f m.
```

(=$_i$ is the SSReflect definition of set equality; ~: is a SSReflect notation for set complementation.)

Using the fact that the probability of a union is smaller that the sum of the probabilities, the left-handside of the conclusion of the main lemma can be bounded by the following expression:

```
Σ_(f : encT A M n)
   Wght.d P f * Pr (W^n (| f 0)) (~: ℬ f 0) +      (* (1) *)
Σ_(i | i ≠ 0) Σ_(f : encT A M n)
   Wght.d P f * Pr (W^n (| f 0)) (ℬ f i)           (* (2) *)
```

The first summand (1) can be rewritten into

```
Pr (𝒥(P , W)^n) (~: 𝒥𝒯𝒮 P W n ε0).
```

which can be bounded using the lemma JTS_1 (Sect. 5.4). The second summand (2) can be rewritten into

```
k * Pr ((P^n) × ((𝒪( P , W ))^n)) [set x | x ∈ 𝒥𝒯𝒮 P W n ε0].
```

which can be bounded using the lemma non_typical_sequences (Sect. 5.4). The bounds ε_0 and n have been carefully chosen so that the proof can be concluded with symbolic manipulations. See [2] for details.

7 The Converse of the Channel Coding Theorem: Proof Approach

The rest of this article is dedicated to the proof of the strong converse of the channel coding theorem (the difference with the weak version was explained in the introduction—Sect. 1). The proof we formalize is based on the so-called *method of types* [9, 10]. The method of types is an important technical tool in information theory. It is a refinement of the approach of typical sequences (that we saw in Sect. 3 and Sect. 5.4).

Here is the plan for the rest of the article:

1. In Sect. 8, we introduce the basic definitions of the method of types.
2. In Sect. 9, we introduce the notion of information divergence that is used to quantify of how much two distributions differ from each other. This is a preparatory step before applying the method of types.
3. In Sect. 10, we exploit the method of types to find an upper-bound for the success rate of decoding.
4. In Sect. 11, we use the above upper-bound for the success rate of decoding to prove the converse of the channel coding theorem.
5. The proof of the converse of the channel coding theorem actually makes use of a technical lemma whose proof is the goal of Sect. 12. We have isolated this lemma because its proof appeals to several results that are of broader interest for the working information theorist.

8 The Method of Types: Basic Definitions

In this section, we introduce the basic definitions of the method of types (types, joint types, conditional types, and typed codes) and prove their basic properties (mostly cardinality properties). Of course, there is more to the method of types than a set of definitions; we will see for example in Sect. 10.1 that conditional types are used to partition the set of `n.-tuple`'s.

8.1 Types and Typed Tuples

8.1.1 The Set of Types: Definition and Cardinality Properties

Given a `n.-tuple t` over `A` and an element `a ∈ A`, `N(a | t)` denotes the number of occurrences of `a` in `t`. Formally, we define it as follows:

```
Definition num_occ a t := count (pred1 a) t.
```

(`pred1 a` is a boolean function that returns `true` for input `a` and `false` otherwise.)

The *type* (or the *empirical distribution*) of a `n.-tuple t` over `A` is the distribution `P` such that `P a` is the relative frequency of `a ∈ A` in `t`, i.e., `P a = N(a | t) / n`. Types are therefore the special case of distributions where all the probabilities are rational numbers with the common denominator `n`. We take advantage of this fact by defining types as the (finite) subtype of distributions with a natural-valued function (with output bounded by `n`) that gives the numerators of the probabilities:

```
Record type : predArgType := mkType {
  d :> dist A ;
  f : {ffun A → 'I_n+1} ;
  d_f : ∀ a, d a = (f a) / n }.
```

('I_(n+1) is the set of naturals $\{0, 1, \ldots, n\}$.) $\mathcal{P}_n$(A) denotes the (finite) type of types corresponding to n.-tuples over A. We introduce a coercion so that a type is silently treated as its underlying distribution (denoted by d in the definition above) when this is required by the context. For example, we can write $\mathcal{H}$P for the entropy of a type P.

Note that $\mathcal{P}_n$(A) is empty when A is empty or when n is 0. Otherwise, the set of types is not empty and its cardinal is upper-bounded as follows:

```
Lemma type_not_empty n : 0 < |A| → 0 < |P_{n+1}(A)|.
Lemma type_counting : |P_n(A)| ≤ (n+1)^|A|.
```

8.1.2 The Set of Typed Tuples: Definition and Basic Properties

Given a type P : $\mathcal{P}_n$(A), we are also interested in the set of all the tuples that have type P. Let us denote this set by T_{P}, defined formally as follows:

```
Definition typed_tuples :=
  [set t : n.-tuple A | [∀ a, P a = N(a | t) / n] ].
```

For a type P : $\mathcal{P}_n$(A), T_{P} is never empty:

```
Lemma typed_tuples_not_empty P : {t | t ∈ T_{P}}.
```

Drawing n times independently with distribution P from a finite set A, the probability of obtaining the tuple t depends only on how often the various elements of A occur in t. In particular with P : $\mathcal{P}_n$(A), we have:

```
Lemma tuple_dist_type t : t ∈ T_{P} →
  P^n t = Π_(a : A) P a ^ (type.f P a).
```

(type.f is the field f of the Record type defined in Sect. 8.1.1.) As a direct consequence, the probability of obtaining a typed tuple is constant and can be expressed succinctly using the entropy of the type (seen as a distribution):

```
Lemma tuple_dist_type_entropy t : t ∈ T_{P} →
  P^n t = exp (- n * ℋ P).
```

From this, we can derive an upper-bound for the cardinal of typed tuples with type P : $\mathcal{P}_n$(A):

```
Lemma card_typed_tuples : | T_{ P } | ≤ exp (n * ℋ P).
```

Observe that the set of typed tuples T_{P} is a subset of the set of perfect typical sequences (we saw typical sequences in Sect. 3):

```
Lemma typed_tuples_are_typ_seq : T_{ P } ⊆ 𝒯𝒮 P n 0.
```

This fact coupled with the lemma TS_sup from Sect. 3.2 provides an alternative way to prove card_typed_tuples.

8.2 Joint Types and Conditional Types

8.2.1 Joint Types: Definition and Cardinality Properties

Given two finite sets `A` and `B`, the *joint type* of a pair of `n.-tuples ta` over `A` and `tb` over `B` is the type of the sequence `zip ta tb` over $(A \times B)^n$, i.e., the channel `W` such that for all `a` $\in$ `A` and `b` $\in$ `B`, `W a b = N( (a, b) | zip_tuple ta tb ) / n` (where `zip_tuple` is the standard zip function of functional programming adapted to tuples). Hereafter, we denote `N( (a, b) | zip_tuple ta tb )` by `N(a, b | ta, tb)`.

Similarly to types that specify the relative frequency of elements in a tuple, *joint types* specify the relative frequency of pairs in a tuple of pairs. Also, similarly to types that are a special case of distributions, joint types can be seen as a special case of channels. In fact, joint types will be used as a way to approximate channels in the proof of the converse of the channel coding theorem. We therefore define joint types as a (finite) subtype of channels with a natural-valued function that gives the relative frequency of pairs of elements:

```
Record jtype : predArgType := mkJtype {
  c :> 𝒞ℋ_1^*(A, B) ;
  f : {ffun A → {ffun B → 'I_n+1}} ;
  sum_f : Σ_(a in A) Σ_(b in B) f a b = n ;
  c_f : ∀ a b, c a b = let row := Σ_(b in B) f a b in
                       if row = 0
                       then 1 / |B|
                       else (f a b) / row }.
```

The channel `c` is defined from the function `f` in such a way that each row of the stochastic matrix is indeed a distribution (and so that the channel is well-defined). We introduce a coercion so that a joint type is silently treated as its underlying channel when this is required by the context. Let $\mathcal{P}_n$`(A , B)` denote the type of joint types.

The cardinal of the set of joint types $\mathcal{P}_n$`(A , B)` is bounded as follows:

```
Lemma jtype_not_empty : 0 < |A| → 0 < |B| → 0 < |𝒫_n (A , B)|.
Lemma bound_card_jtype : |𝒫_n(A , B)| ≤ (n+1)^(|A| * |B|).
```

8.2.2 Shells and Conditional Types: Definition and Cardinality Properties

Let us assume the joint type `V` : $\mathcal{P}_n$`(A , B)`. We say that the n-tuple `tb` has type `V` given the n-tuple `ta` when for all `a`, `b`, we have `(jtype.f V) a b = N(a, b | ta, tb)` (recall that `jtype.f` is the natural-valued function from the definition of joint types—Sect. 8.2.1). `V.-shell ta` denotes the set of tuples that have type `V` given `ta`. It is called the `V`-*shell* of `ta` and is defined formally as follows:

```
Definition shell :=
  [set tb : n.-tuple B | [∀ a, [∀ b, N(a, b |ta, tb) = (jtype.f V) a b]]].
```

We are given a type `P` : $\mathcal{P}_n$`(A)` and a finite type `B`. The *conditional types* ν`^{B}(P)` are the joint types `V` of type $\mathcal{P}_n$`(A , B)` such that for all the typed tuples `ta` $\in$ `T_{P}`, the shell `V.-shell ta` is not empty:

```
Definition cond_type :=
  [set V : 𝒫_n ( A , B ) | [∀ ta, (ta ∈ T_{P}) ⇒ (V.-shell ta ≠ ∅ )]].
```

So, by definition, the shells of typed tuples are never empty for conditional types. This condition is actually equivalent to the following one:

```
Definition row_num_occ (V : 𝒫_n (A , B)) := ∀ ta, ta ∈ T_{P} →
  ∀ a, Σ_(b in B) (jtype.f V) a b = N(a | ta).
```

We often resort to this equivalent condition in our technical developments.

For a conditional type `V ∈ ν^{B}(P)`, the size of the shells of typed tuples `ta ∈ T_{P}` is upper-bounded as follows:

```
Lemma card_shelled_tuples : | V.-shell ta | ≤ exp (n * ℋ(V | P )).
```

8.3 Typed Codes: Definitions

A *typed code* is a code whose codewords are tuples typed w.r.t. some type `P : 𝒫_n(A)`, i.e., they all belong to `T_{P}`:

```
Record typed_code := mkTypedCode {
  untyped_code :> code A B M n ;
  typed_prop : ∀ m, enc untyped_code m ∈ T_{P} }.
```

Given a type `P` and a code `c`, there is a canonical way to construct a typed code. We denote by `P.-typed_code` of `c` the *typed code of* `c` defined as follows. First, we pick up a typed tuple `def ∈ T_{P}`. This is always possible because we have proved that `T_{P}` is never empty (Sect. 8.1.2):

```
Definition def := sval (typed_tuples_not_empty P).
```

(`sval` retrieves the witness from a constructive proof of existence.) The `P.-typed_code` of `c` is the same code as `c` except that the encoding function returns `def` when `enc c m` is not typed by `P`:

```
Definition tcode_untyped_code := mkCode
  [ffun m ⇒ if enc c m ∈ T_{P} then enc c m else def] (dec c).
Lemma tcode_typed_prop : ∀ m : M, (enc tcode_untyped_code) m ∈ T_{P}.
Definition tcode : typed_code B M P := mkTypedCode tcode_typed_prop.
```

Typed codes will be used in Sect. 10.1.

9 Information Divergence

The *information divergence* (or *relative entropy*) is a measure of the difference between two distributions. Let `𝒟(P || Q)` denote the information divergence between two distributions `P` and `Q`. It is defined as follows:

```
Definition div P Q := Σ_(a in A) P a * (log (P a) - log (Q a)).
```

In information theory, the information divergence can be infinite. This does not happen as long as `P` is *dominated* by `Q`, i.e., when, for any `a ∈ A`, `Q a = 0` implies `P a = 0`. Hereafter, `P ≪ Q` denotes the fact that `P` is dominated by `Q`, which is formally defined as follows:

```
Definition dom_by (P Q : A → R) := ∀ a, Q a = 0 → P a = 0.
```

Under the hypothesis `P ≪ Q`, we can prove for example that the divergence `𝒟(P || Q)` is positive:

```
Lemma leq0div : 0 ≤ 𝒟(P || Q).
```

Given a distribution P and two channels V and W of type $\mathcal{CH}_1$(A, B), $\mathcal{D}$(V || W | P) denotes the *conditional divergence* defined as follows:

```
Definition cdiv V W P := Σ_(a : A) P a * 𝒟(V a || W a).
```

Like the information divergence, the conditional divergence can be infinite. This does not happen as long as, for any a ∈ A such that P a ≠ 0, V a is dominated by W a. We denote this property by V ≪ W | P, which is formally defined as follows:

```
Definition cdom_by V W P := ∀ a, P a ≠ 0 → (V a) ≪ (W a).
```

Under the hypothesis V ≪ W | P, we can for example prove that the conditional divergence $\mathcal{D}$(V || W | P) is positive:

```
Lemma leq0cdiv : 0 ≤ 𝒟(V || W | P).
```

Our formal definition of the conditional divergence $\mathcal{D}$(V || W | P) does not agree with the standard mathematical definition when V ≪ W | P does not hold: we do not model the fact that it should be infinite. However, it will turn out in the following that the conditional divergence is always wrapped into an exponential. Therefore, instead of resorting to extended reals, we introduce the following definition:

```
Definition exp_cdiv P V W := if V ≪ W | P
                             then exp (- n * 𝒟(V || W | P))
                             else 0.
```

Here, P is a type, V is a conditional type, and W is a channel.

We now prove two technical results about the conditional divergence. Let us consider two channels V and W of type $\mathcal{CH}_1$(A, B) and a distribution P such that V ≪ W | P. It happens that the conditional divergence can also be expressed as the information divergence of the joint distributions:

```
Lemma cdiv_is_div_joint_dist : 𝒟(V || W | P) =  𝒟(𝒥(P , V) || 𝒥(P , W)).
```

Let us consider a channel W of type $\mathcal{CH}_1$(A, B), a type P : $\mathcal{P}_n$(A), and a conditional type V ∈ ν^{B}(P). Assume that V ≪ W | P. Then, the probability of receiving tb when ta is sent over a DMC with the probability transition matrix W is:

```
Lemma dmc_cdiv_cond_entropy :
  W^n (tb | ta) = exp (- n * (𝒟(V || W | P) + ℋ(V | P))).
```

The conditional divergence is here wrapped into an exponential, so that this lemma can be rephrased equivalently using the definition exp_cdiv introduced above:

```
Lemma dmc_exp_cdiv_cond_entropy :
  W^n (tb | ta) = exp_cdiv P V W * exp (- n * ℋ(V | P)).
```

10 The Method of Types: Upper-bound for the Success Rate of Decoding

In Sect. 6.1, we defined the error rate $\bar{e}_{cha}$(W, c) of a code c over a channel W. The *success rate* is defined similarly:

```
Definition s̄_cha (W : 𝒞ℋ_1(A, B)) (c : code A B M n) := 1 - ē_cha(W , c).
```

The goal of this section is to establish a suitable upper-bound of the success rate $\bar{s}_{cha}$(W, c) for any code c. This is where the method of types is put at work. Let us provide some intuition about the organization of this section. We will first go through the intermediate step of finding an upper-bound of $\bar{s}_{cha}$(W, tc) in the special case where tc is a typed code (typed codes have been defined in Sect. 8.3). In the case of a typed code tc, we will see that $\bar{s}_{cha}$(W, tc) can be bounded by an expression of the form $(n+1)^{|A||B|}\exp(-nL)$ where L is some positive value (Sect. 10.1). Furthermore, we will see that, for any code c, there is a relation between the success rate for c and the success rate for a typed version of c such that $\bar{s}_{cha}$(W, c) can be bounded by an expression of the form $(n+1)^{|A|+|A||B|}\exp(-nL)$ (Sect. 10.2). Then it will become possible to show why the success rate of decoding can be made arbitrarily small so as to prove the converse of the channel coding theorem (Sect. 11).

10.1 Success Rate for Typed Codes

In order to upper-bound the success rate for typed codes, we express it as a sum over conditional types. This partitioning is actually typical of the method of types; we will therefore take some time to detail formal proofs in this section.

10.1.1 Partitioning using Types

We consider a code tc with the set of messages M such that tc is typed w.r.t. P of type $\mathcal{P}_n$(A). We will express the success rate of tc as a sum over conditional types, more precisely, using traditional mathematical notations, as follows:

$$\bar{s}_{cha}(W,tc) = \sum_{V\in\nu^B(P)} \exp(-nD(V||W|P)) \underbrace{\frac{\exp(-n\mathcal{H}(V|P))}{|M|}\sum_{m\in M}|T_V((enc\ tc)(m))\cap(dec\ tc)^{-1}(m)|}_{\texttt{success\_factor}\ tc\ V}$$

Each term of this sum is expressed using an expression that we abbreviate as success_factor tc V (where V is a conditional type):

```
Definition success_factor tc V :=
  exp (- n * ℋ(V | P)) / |M| *
  Σ_ (m : M) | (V.-shell ((enc tc) m )) ∩ ((dec tc) @^-1: [set Some m]) |.
```

(f @^-1: A is the preimage of A under f.) Using this definition and expressing the exponential of the conditional divergence using exp_cdiv from the previous section (Sect. 9), we prove that the success rate of a typed code tc typed w.r.t. P can be written as follows:

```
Lemma typed_success tc : s̄_cha(W, tc) =
  Σ_ (V | V ∈ ν^{B}(P)) exp_cdiv P V W * success_factor tc V.
```

Proof First, we express $\bar{s}_{cha}$(W, tc) in terms of the arithmetic average of the probability of decoding success knowing that a particular message m was sent. This is made possible by the following lemma, that follows directly from the definition

of the error rate (Sect. 6.1), and that holds independently of whether the code is typed or not:

```
Lemma success_decode (W : CH_1(A, B)) (c : code A B M n) :
  s̄_cha(W, c) = 1 / |M| *
    Σ_(m : M) Σ_(tb | dec c tb = Some m) (W^n (| enc c m)) tb.
```

Our goal can therefore be expressed as an equality between two sums over messages `M`. We rearrange our goal on both sides of the equality so as to put `Σ_(m : M)` the most outwards, and then look at the summands below `Σ_(m : M)`. Our goal now boils down to check whether the following holds for any `m`:

```
Σ_(tb | (dec tc) tb = Some m) W^n (tb | ((enc tc) m)) =
Σ_(V | V ∈ ν^{B}(P))
   exp_cdiv P V W *
   | V .-shell ((enc tc) m) ∩ dec tc @^-1: [set Some m] | *
   exp (- n * H(V | P))
```

The left-handside is a sum over the tuples `tb` of type `n.-tuple B`. It happens that the latter set can be partitioned according to conditional types. More precisely, given a tuple `ta` of type `P : P_n(A)` and a finite set `B`, the set of shells `V.-shell ta` where `V` is a conditional type `ν^{B}(P)` partitions the set `n.-tuple B`. The partition in question is the set of sets formally written `(fun V ⇒ V.-shell ta) @: ν^{B}(P)` (`f @: A` is the image of `A` by `f`). Using the fact that the latter set of sets is indeed a partition, we prove the following lemma:

```
Lemma sum_tuples_ctypes f F :
  Σ_(tb | F tb) f tb =
  Σ_(V | V ∈ ν^{B}(P)) Σ_ (tb in V.-shell ta | F tb) f tb.
```

We use this lemma to change the summation `Σ_(tb | ...)` on the left-handside of our goal to a summation `Σ_(V | V ∈ ν^{B}(P))` and simplify our goal by looking at the summands on both sides. For any message `m` and conditional type `V`, we now want to prove that:

```
exp (- n * H(V | P)) * exp_cdiv P V W = W^n (tb | ((enc tc) m))
```

This actually corresponds to lemma `dmc_exp_cdiv_cond_entropy` that we saw at the end of Sect. 9.

10.1.2 Bounding Using Types

In the previous section, we rewrote the success rate of decoding for a typed code as a sum over conditional types. Each summand was expressed using `success_factor` and what we prove now is that the latter can actually by upper-bounded as follows:

```
Definition success_factor_bound M V P :=
  exp(- n * |+ log (|M|) / n - I(P ; V) |).
Lemma success_factor_ub M V P tc :
  success_factor tc V ≤ success_factor_bound M V P.
```

`|+r|` is `r` if `r` is positive and `0` otherwise. The joint type `V` is actually a conditional type (i.e., `V ∈ ν^{B}(P)`).

Proof When `log |M| / n < I(P ; V)`, it is sufficient to prove:

```
Lemma success_factor_bound_part1 : success_factor tc V ≤ 1.
```

This is a consequence of the lemma `card_shelled_tuples` (Sect. 8.2.2).

In the case where `log |M| / n ≥ 𝓘(P ; V)`, we rewrite the goal as follows:

```
success_factor tc V ≤ exp (n * 𝓘(P ; V)) / |M|
```

which simplifies to:

```
Σ_(m : M) |(V .-shell ((enc tc) m)) ∩ dec tc @^-1: [set Some m]| ≤
  exp (n * ℋ(P ∘ V))
```

Under the hypothesis `V ∈ ν^{B}(P)`, it happens that `ℋ(P ∘ V) = ℋ(𝒪(V))` where `𝒪(V)` is the type of type $\mathcal{P}_n$`(B)` corresponding to the following distribution (defined in a module `OutType`):

```
Definition f := fun b ⇒ (Σ_(a in A) (jtype.f V) a b) / n.
Lemma f0 (b : B) : 0 ≤ f b.
Lemma f1 : Σ_(b in B) f b = 1.
Definition d : dist B := makeDist f0 f1.
```

Using the lemma `card_typed_tuples` (Sect. 8.1.2), we can therefore change the goal to:

```
Σ_(m : M) |(V .-shell ((enc tc) m)) ∩ dec tc @^-1: [set Some m]| ≤
  |T_{𝒪(V)}|
```

We observe that `V`-shells are subsets of the tuples typed by the output type, i.e., for any tuple `ta`, we have:

```
V.-shell ta ⊆ T_{ 𝒪(V) }.
```

The goal can therefore be changed as follows:

```
Σ_(m : M) |T_{𝒪(V)} ∩ dec tc @^-1: [set Some m]| ≤ |T_{𝒪(V)}|
```

Since the sets `T_{𝒪(V)}∩(dec tc @^-1: [set Some m])` are pairwise disjoint, the sum of cardinals can be changed to the cardinal of the union:

```
|⋃_(m : M) T_{𝒪(V)} ∩ (dec tc @^-1: [set Some m])| ≤ |T_{𝒪(V)}|
```

This concludes the proof because we are comparing the cardinal of a set with the cardinal of a union of its subsets.

The lemma above gives an upper-bound for `success_factor tc V`. From this lemma, we can derive an upper-bound for the success rate $\bar{s}_{cha}$`(W, tc)`, because the latter can be expressed using `success_factor tc V` (previous section, Sect. 10.1.1).

Let `V0` be some joint type $\mathcal{P}_n$`(A , B)`. Such a joint type always exists since the set of joint types is never empty (Sect. 8.2.1). Let `Vmax` be the argument of the maximum of the function `exp_cdiv_bound` defined as follows:

```
fun V ⇒ exp_cdiv P V W * success_factor_bound M V P
```

Then the success rate for a typed code `tc` typed w.r.t. `P` can be upper-bounded as follows:

```
Lemma typed_success_bound :
  let Vmax := arg_rmax V0 [pred V | V ∈ ν^{B}(P)] exp_cdiv_bound in
  s̄_cha(W, tc) ≤ (n+1)^(|A| * |B|) * exp_cdiv_bound Vmax.
```

10.2 Upper-bound of the Success Rate for any Code

We can use the upper-bound of the success rate for typed codes to upper-bound the success rate for any code.

Let `c` be some code. Let `Pmax` by the type that maximizes the success rate of the typed codes of `c` (the typed codes of a code were defined in Sect. 8.3). Let `P0` be some default type that we know always exists because the set of types is never empty (Sect. 8.1.1). The success rate of the code `c` can be upper-bounded as follows:

```
Lemma success_bound :
  let Pmax := arg_rmax P0 predT (fun P ⇒ s̄_cha(W, P.-typed_code c)) in
  s̄_cha(W, c) ≤ (n+1)^|A| * s̄_cha(W, Pmax.-typed_code c).
```

Proof Using the lemma `type_counting` (Sect. 8.1.1), we rewrite our goal as follows:

```
s̄_cha(W, c) ≤ |P_n(A)| * s̄_cha(W, Pmax.-typed_code c)
```

or, equivalently, if we express the cardinal as a sum:

```
s̄_cha(W, c) ≤ Σ_(P : P_n(A)) s̄_cha(W, P.-typed_code c)
```

We change the expression of the success rate on the left-handside of our goal by using the lemma `success_decode` from the previous section (Sect. 10.1.1):

```
1 / |M| * (Σ_(m : M) Σ_(tb | dec c tb = Some m) W^n (tb | enc c m)) ≤
  Σ_(P : P_n(A)) s̄_cha(W, P.-typed_code c)
```

The left-handside is a sum over the messages `m` of type `M`. It happens that the latter set can be partitioned according to types. More precisely, given a code `c`, the set of non-empty sets of messages `[set m | enc c m ∈ T_{P}]`, with `P` ranging over types, partitions the set of messages `M`. We define the sets of messages above as follows:

```
Definition enc_pre_img c (P : P_n(A)) := [set m | enc c m ∈ T_{P}].
```

The partition in question is the set of sets formally written as follows:

```
Definition enc_pre_img_partition :=
  enc_pre_img @: [set P in P_n(A) | enc_pre_img P ≠ ∅ ].
```

(Recall that `f @: A` is the image of `A` by `f`.) Using the fact that the latter is indeed a partition, we prove the following lemma:

```
Lemma sum_messages_types f : Σ_(m : M) (f m) =
  Σ_(P : P_n(A)) Σ_(m | m ∈ enc_pre_img c P) f m.
```

We use this last lemma to rewrite the left-handside of our goal so that it reduces to check that we have, for any `P : P_n(A)`:

```
1 / |M| * (Σ_(m | m ∈ enc_pre_img c P)
             Σ_(tb | dec c tb = Some m) W^n (tb | enc c m)) ≤
  s̄_cha(W, P.-typed_code c)
```

We again use the lemma `success_decode` from the previous section (Sect. 10.1.1) but this time to change the expression of the success rate on the right-handside:

```
Σ_(m | m ∈ enc_pre_img c P) Σ_(tb | dec c tb = Some m)
  W^n (tb | enc c m) ≤
Σ_(m : M) Σ_(tb | dec (P.-typed_code c) tb = Some m)
  W^n (tb | enc (P.-typed_code c) m)
```

We observe that, when `m ∈ enc_pre_img c P`, we have the following equality:

```
W^n (tb | enc c m) = W^n (tb | enc (P.-typed_code c) m)
```

This observation coupled with the fact that `dec c` is identical to `dec (P.-typed_code c)` means that the left-handside can be equivalently expressed using `dec (P.-typed_code c)`. This concludes the proof because the left-handside is a sum over a subset of the right-handside.

11 The Channel Coding Theorem—Converse Part

11.1 Channel Coding Theorem—Statement of the Converse Part

Given a discrete channel `W` of type $\mathcal{CH}_1{}^*$`(A, B)` with capacity `cap`, the converse of the channel coding theorem says that any code whose code rate is greater than the capacity `cap` has negligible success rate. Formally, let `minRate` be a rate greater than the capacity (`minRate > cap`). Then, for any ε such that `0 < ε`, there is a block length n_0 above which the success rate of any code with rate greater than `minRate` is smaller than ε:

```
Theorem channel_coding_converse : ∃ n₀,
  ∀ n M (c : code A B M n),
    0 < |M| → n₀ < n → minRate ≤ CodeRate c → s̄_cha(W, c) < ε.
```

11.2 Channel Coding Theorem—Proof of the Converse Part

We prove the converse of the channel coding theorem using an intermediate, more general lemma. We are given a channel `W` of type $\mathcal{CH}_1{}^*$`(A, B)` with capacity `cap`. We show that, for any `minRate > cap`, there is a strictly positive Δ such that the success rate can be upper-bounded as follows:

```
Lemma channel_coding_converse_gen : ∃ Δ, 0 < Δ ∧ ∀ n',
  let n := n'+1 in ∀ M (c : code A B M n), 0 < |M| →
    minRate ≤ CodeRate c →
      s̄_cha(W, c) ≤ (n+1)^(|A| + |A| * |B|) * exp (- n * Δ).
```

Proof First, we instantiate the existential variable with a strictly positive Δ that is "small enough", i.e., such that, for any distribution `P` and channel `V` satisfying `V ≪ W | P`, we have:

```
Δ ≤ 𝒟(V || W | P) + |⁺ minRate - ℐ(P ; V) |      (* (3) *)
```

The existence of such a strictly positive real Δ is established by an intermediate lemma (namely, `error_exponent_bound`) whose proof is deferred to Sect. 12.4 because it requires new concepts and technical lemmas. Indeed, to make sure that such a strictly positive Δ exists, we investigate in particular the situation where $\mathcal{D}$`(V || W | P)` is 0. For this purpose, it is better to be able to evaluate numerically the resemblance between `V` and `W`. Unfortunately, the value of the information divergence is not a good indication because it is not a distance. This is why we will introduce the *variation distance* and *Pinsker's inequality* in Sect. 12.2.

We use the lemma `success_bound` from the previous section (Sect. 10.2) to upper-bound $\bar{s}_{cha}$`(W, c)` to transform our goal to:

$\bar{s}_{cha}$(W, tc) ≤ (n+1)^(|A| * |B|) * exp (- n * Δ)

where tc is the Pmax.-typed_code of c with Pmax the argument of the maximum as defined in Sect. 10.2.

Now that we have made appear the success rate of the typed code tc, we can upper-bound $\bar{s}_{cha}$(W, tc) using the lemma typed_success_bound (Sect. 10.1.2) and transform our goal to:

exp_cdiv Pmax Vmax W * success_factor_bound M Vmax Pmax ≤ exp (- n * Δ)

where Vmax is the argument of the maximum as defined in Sect. 10.1.2.

We develop the expressions of exp_cdiv and success_factor_bound to simplify our goal to:

exp (- n * 𝒟(Vmax || W | Pmax)) *
  exp (- n * |⁺ log |M| / n - ℐ(Pmax ; Vmax) |) ≤ exp (- n * Δ)

The conclusion follows from the upper-bound of Δ given by (3).

Proof (of the converse of the channel coding theorem) We use the intermediate lemma channel_coding_converse_gen proved just above to obtain Δ such that, for any code c of code rate larger than minRate, we have:

$\bar{s}_{cha}$(W, c) ≤ (n+1)^(|A| + |A| * |B|) * exp (- n * Δ) (* (4) *)

We instantiate n_0 in the statement of the converse of the channel coding theorem with 2^K * (K+1)! / (Δ * ln 2)^(K+1) / ε where K is |A| + |A| * |B|. Using (4), the goal then boils down to:

(n+1)^K * exp (- n * Δ) < ε

for any n such that n_0 < n. The conclusion follows from symbolic manipulations using the following lower-bound of the exponential function:

```
Lemma exp_lb n x : 0 ≤ x → x^n / n! ≤ exp x.
```

12 Error Exponent Bound for the Channel Coding Theorem

The goal of this section is to prove the lemma error_exponent_bound that we used in the proof of the converse of the channel coding theorem (Sect. 11.2). We provide this proof for the sake of completeness and isolate it in this section because it is technical and requires intermediate lemmas that are of general interest for the working information theorist. For example, the log-sum inequality (Sect. 12.1) is used to prove various convexity results [8, Sect. 2.7] and the variation distance (Sect. 12.2) has further applications in statistics [8, Sect. 11].

12.1 The log-sum Inequality

Let f and g be two positive real-valued functions with domain A such that f is dominated by g. Let also C be a subset of A. To simplify the presentation in this section, we denote Σ_(a | a ∈ C) f a by Σ_{C} f (and similarly for other sets and functions). The log-sum inequality is the following inequality:

```
Definition log_sum_stmt A (C : {set A}) (f g : A → R+) :=
  f ≪ g →
  Σ_{C} f * log (Σ_{C} f / Σ_{C} g) ≤ Σ_(a | a ∈ C) f a * log (f a / g a).
Lemma log_sum A (C : {set A}) (f g : A → R+) : log_sum_stmt C f g.
```

Proof Let `D` be the subset of `C` on which `f` is strictly positive, i.e., `D` is defined as `[set a | a ∈ C ∧ f a ≠ 0]`. We observe on the one hand that:

```
Σ_{C} f * log (Σ_{C} f / Σ_{C} g) ≤ Σ_{C} f * log (Σ_{C} f / Σ_{D} g)
```

and on the other hand that:

```
Σ_(a | a ∈ D) f a * log (f a / g a) ≤ Σ_(a | a ∈ C) f a * log (f a / g a)
```

To prove the log-sum inequality it is thus sufficient to prove the following inequality:

```
Σ_{D} f * log (Σ_{D} f / Σ_{D} g) ≤ Σ_(a | a ∈ D) f a * log (f a / g a).
```

Since the latter is the log-sum inequality in the special case where `f` is strictly positive, the log-sum inequality can therefore be derived from the following lemma:

```
Lemma log_sum1 {A : finType} (C : {set A}) (f g : A → R+) :
  (∀ a, a ∈ C → 0 < f a) → log_sum_stmt C f g.
```

To prove log_sum1, we can actually make a number of simplifying assumptions. We can assume that `C` is not empty, that `g` is never 0, and that Σ`_{C} f` and Σ{C} `g` are not 0. We can even assume without loss of generality that Σ`_{C} f = `Σ`_{C} g`. Indeed, suppose that log_sum1 holds when Σ`_{C} f = `Σ`_{C} g`. Then, for the general case, we define `k` as Σ`_{C} f / `Σ`_{C} g`, so that the conclusion of log_sum1 amounts to:

```
Σ_{C} f * log k ≤ Σ_(a | a ∈ C) f a * log (f a / g a)    (* (5) *)
```

Since Σ`_(a | a ∈ C) k * g a = `Σ`_{C} f`, we can use the without-loss-of-generality hypothesis to derive:

```
Σ_{C} f * log (Σ_{C} f / Σ_(a | a ∈ C) k * g a) ≤
  Σ_(a | a ∈ C) f a * log (f a / (k * g a))
```

whose left-handside is equal to `0`, so that it can be rewritten as:

```
0 ≤ Σ_(a | a ∈ C) f a * log (f a / g a) - f a * log k
```

from which the inequality (5) is a consequence.

Let us now prove log_sum1 in the particular case where Σ`_{C} f = `Σ`_{C} g`, i.e.:

```
0 ≤ Σ_(a | a ∈ C) f a * log (f a / g a)
```

This can be equivalently rewritten as follows, using the natural logarithm `ln`:

```
Σ_(a | a ∈ C) f a * (1 - g a / f a) ≤
  Σ_(a | a ∈ C) f a * ln (f a / g a)
```

which is a consequence of the following lemma, whose proof follows by a standard analytic argument:

```
Lemma ln_id_cmp : ∀ x, 0 < x → ln x ≤ x - 1.
```

12.2 Variation Distance and Pinsker's Inequality

As we hinted at when proving the converse of the channel coding theorem (see Sect. 11.2), it is useful to introduce, in addition to the information divergence, a distance to evaluate numerically the resemblance between two distributions. We denote the *variation distance* of two distributions `P` and `Q` by `d(P, Q)`, defined formally as follows:

```
Definition var_dist (P Q : dist A) := Σ_(a : A) Rabs (P a - Q a).
```

Pinsker's inequality establishes a relation between the variation distance and the information divergence. It shows that when the information divergence is close to 0, then so is the variation distance, and therefore the two distributions are similar.

To prove Pinsker's inequality, we will first prove the special case of distributions over sets with two elements (Sect. 12.2.2) and then generalize this result to any distribution (Sect. 12.2.3). This generalization is achieved using an intermediate lemma called the *partition inequality* (Sect. 12.2.1).

12.2.1 The Partition Inequality

Before proving Pinsker's inequality, we prove an intermediate lemma that relates the information divergence of any two distributions `P` and `Q` with distributions over the set of booleans built after `P` and `Q`.

We consider a finite set `A` and a partition of it into two sets `A_0` and `A_1`. In other words, we have the following hypotheses: `A_0 ∩ A_1 = ∅` and `A_0 ∪ A_1 = [set: A]`. Let `P` and `Q` be two distributions over `A`. Using `P` and `A`, we construct a distribution `P_A` that associates to the boolean `false` (denoted by `0`) the sum `Σ_(a in A_0) P a` and to the boolean `true` (denoted by `1`) the sum `Σ_(a in A_1) P a`. `Q_A` is constructed similarly. Then, when `P ≪ Q` holds, we have:

```
Lemma partition_inequality : 𝒟(P_A || Q_A) ≤ 𝒟(P || Q).
```

Proof We develop the right-handside `𝒟(P || Q)` to obtain the goal:

```
𝒟(P_A || Q_A) ≤ Σ_(a | a ∈ A_0) P a * log (P a / Q a) +
                Σ_(a | a ∈ A_1) P a * log (P a / Q a)
```

We use the log-sum inequality (Sect. 12.1) to further low down the right-handside and obtain, after simplification:

```
𝒟(P_A || Q_A) ≤ P_A 0 * log (P_A 0 / Q_A 0) + P_A 1 * log (P_A 1 / Q_A 1)
```

We now develop the left-handside `𝒟(P_A || Q_A)` to obtain the goal:

```
P_A 0 * (log (P_A 0) - log (Q_A 0)) + P_A 1 * (log (P_A 1) - log (Q_A 1)) ≤
  P_A 0 * log (P_A 0 / Q_A 0) + P_A 1 * log (P_A 1 / Q_A 1)
```

Finally, we perform a case analysis w.r.t. `P_A 0`, `Q_A 0`, `P_A 1`, and `Q_A 1`. Suppose for example that `P_A i > 0` and `Q_A i > 0` for any `i` $\in \{0, 1\}$, then the goal can be simplified to:

```
P_A 0 * (log (P_A 0) - log (Q_A 0)) + P_A 1 * (log (P_A 1) - log (Q_A 1)) ≤
  P_A 0 * (log (P_A 0) - log (Q_A 0)) + P_A 1 * (log (P_A 1) - log (Q_A 1))
```

that trivially holds. The other cases where some of the `P_A i` or `Q_A i` are 0 are similar, possibly resorting to the hypothesis `P ≪ Q` to establish a contradiction.

12.2.2 Pinsker's Inequality for Distributions over Sets with Two Elements

Let us first prove Pinsker's inequality in the special case of distributions over sets with two elements. We are given a finite type `A` with only two elements `a_0` and `a_1`. Let `p` be a probability and `P` be the distribution such that `P a_0` is `1 - p` and `P a_1` is `p`. Using the probability `q`, we construct the distribution `Q` similarly to `P`. If we assume that `P ≪ Q`, then we have:

```
Lemma Pinsker_2_inequality : 1 / (2 * ln 2) * d(P , Q)^2 ≤ 𝒟(P || Q).
```

Proof We consider the following function:

$$q \mapsto p \log \frac{p}{q} + (1-p) \log \frac{1-p}{1-q} - 4c(p-q)^2.$$

Using the standard Coq library for reals, this function can be written as the following Coq function `pinsker_fun` with parameters `p`, `c`, and `q`:

```
Definition pinsker_fun p c := fun q ⇒
  p * log (div_fct (fun _ ⇒ p) id q) +
  (1 - p) * log (div_fct (fun _ ⇒ 1 - p) (fun q ⇒ 1 - q) q) -
  4 * c * comp (fun x ⇒ x^2) (fun q ⇒ p - q) q.
```

It happens that the lemma we are trying to prove is in fact:

```
0 ≤ pinsker_fun p (1 / (2 * ln 2)) q
```

The latter inequality derives from the following inequality, proved by analyzing the function `pinsker_fun`:

```
Lemma pinsker_fun_pos c : 0 ≤ c ≤ 1 / (2 * ln 2) → 0 ≤ pinsker_fun p c q.
```

12.2.3 Pinsker's Inequality in the General Case

We now generalize Pinsker's inequality for distributions over sets with two elements (Sect. 12.2.2) to any distribution using the partition inequality (Sect. 12.2.1).

Pinsker's inequality in the general case can be stated as follows. For two distributions `P` and `Q` such that `P ≪ Q`, the following inequality holds:

```
Lemma Pinsker_inequality : 1 / (2 * ln 2) * d(P , Q)^2 ≤ 𝒟(P || Q).
```

Proof We consider the partition of the set `A` into the two sets `[set a | Q a ≤ P a]` and `[set a | P a < Q a]`. Based on this partition, we use the partition inequality (lemma `partition_inequality`, Sect. 12.2.1) with the distributions `P` and `Q` to obtain the following inequality:

```
𝒟(P_A || Q_A) ≤ 𝒟(P || Q)
```

Then, Pinsker's inequality can actually be reformulated as follows:

```
1 / (2 * ln 2) * d(P , Q)^2 ≤ 𝒟(P_A || Q_A)
```

On another hand, we can establish that `d(P , Q) = d(P_A , Q_A)`. In consequence, Pinsker's inequality in the general case derives from the Pinsker's inequality for distributions over sets with two elements (lemma `Pinsker_2_inequality`, Sect. 12.2.2).

12.3 Distance from the Mutual Information of one Channel to Another

The goal of this section is to find an upper-bound for the distance from the mutual information of one channel to another.

First, we establish an upper-bound for the distance from the joint entropy of one channel `V` to another channel `W` of type $\mathcal{CH}_1$ `(A, B)`. Let us assume an input distribution `P` such that `V` $\ll$ `W | P`. Let `xlnx` be the Coq definition of the function $x \mapsto x\ln(x)$. Then, if we assume $\mathcal{D}$`(V || W | P)` $\leq$ `(exp (-2))^2 / 2`, we have:

```
Lemma joint_entropy_dist_ub : Rabs (ℋ(P , V) - ℋ(P , W)) ≤
  1 / ln 2 * |A| * |B| * (- xlnx (sqrt (2 * 𝒟(V || W | P)))).
```

Proof The entropy can be expressed alternatively by using the function `xlnx`:

```
Lemma xlnx_entropy P : ℋ P = 1 / ln 2 * (- Σ_(a : A) xlnx (P a)).
```

We use this alternative expression of the entropy to transform our goal to:

```
Rabs (- Σ_(x : [finType of A * B]) xlnx (𝒥(P , V) x)
      + Σ_(x : [finType of A * B]) xlnx (𝒥(P , W) x)) ≤
  |A| * |B| * (- xlnx (sqrt (2 * 𝒟(V || W | P))))
```

Then, we use the triangular inequality to move the sums the most outwards on the left-handside:

```
Σ_(x : [finType of A * B]) Rabs (- xlnx (V x.1 x.2 * P x.1)
                                 + xlnx (W x.1 x.2 * P x.1)) ≤
  |A| * |B| * (- xlnx (sqrt (2 * 𝒟(V || W | P))))
```

It is now sufficient to prove for any `a` $\in$ `A` and `b` $\in$ `B` that:

```
Rabs (xlnx (W a b * P a) - xlnx (V a b * P a)) ≤
  - xlnx (sqrt (2 * 𝒟(V || W | P)))
```

We can simplify the latter goal by getting rid of logarithms:

```
Rabs (W a b * P a - V a b * P a) ≤ sqrt (2 * 𝒟(V || W | P))
```

thanks to the following property of the `xlnx` function:

```
Lemma Rabs_xlnx a x y : 0 ≤ a ≤ exp (-2) →
  0 ≤ x ≤ 1 → 0 ≤ y ≤ 1 → Rabs (x - y) ≤ a →
  Rabs (xlnx x - xlnx y) ≤ - xlnx a.
```

(This is why the hypothesis $\mathcal{D}$`(V || W | P)` $\leq$ `exp (-2) ^ 2 * / 2` is required.) On another hand, the following relation with the variation distance between the joint distributions holds trivially:

```
Rabs (W a b * P a - V a b * P a) ≤ d(𝒥(P , V), 𝒥(P , W))
```

and we also know that the conditional divergence is equal to the information divergence of the joint distributions (lemma `cdiv_is_div_joint_dist`, Sect. 9). Our goal therefore reduces to:

```
d(𝒥(P , V), 𝒥(P , W)) ≤ sqrt (2 * 𝒟(𝒥(P , V) || 𝒥(P , W)))
```

The latter is proved using Pinsker's inequality, which was the topic of the previous section (Sect. 12.2).

We just found an upper-bound for the distance `Rabs (ℋ(P , V) - ℋ(P , W))` (joint entropy). By a similar argument, we can find a similar upper-bound for the distance `Rabs (ℋ(P ∘ V) - ℋ(P ∘ W))` (output entropy). Both results can be combined to obtain an upper-bound for the distance from the mutual information of one channel to another.

Let us assume two channels `V` and `W` of type `𝒞ℋ₁ (A, B)`. Let `P` be an input distribution such that `𝒟(V || W | P) ≤ (exp (-2))^2 / 2` and `V ≪ W | P`. Then we have:

```
Lemma mut_info_dist_ub : Rabs (ℐ(P ; V) - ℐ(P ; W)) ≤
  1 / ln 2 * (|B| + |A| * |B|) * (- xlnx (sqrt (2 * 𝒟(V || W | P)))).
```

12.4 Error Exponent Bound

We finally prove the existence of a strictly positive lower-bound Δ for use in the proof of the converse of the channel coding theorem (Sect. 11.2):

```
Lemma error_exponent_bound : ∃ Δ, 0 < Δ ∧
  ∀ P : dist A, ∀ V : 𝒞ℋ₁(A, B),
    V ≪ W | P → Δ ≤ 𝒟(V || W | P) + |⁺ minRate - ℐ(P ; V) |.
```

Proof The crux of the proof is to carefully instantiate Δ such that one can exploit the upper-bound found in the previous section (Sect. 12.3). Concretely, we first instantiate Δ with `min((minRate - cap) / 2, x^2 / 2)` where `x` is a strictly positive real such that:

```
- xlnx x < min(exp (-2), gamma)      (* (6) *)
```

where `gamma` is defined as:

```
1 / (|B| + |A| * |B|) * ((ln 2) * (minRate - cap) / 2)
```

(We use the continuity of `xlnx` to obtain such an `x`.)

Then, let us assume (otherwise, the conclusion follows easily) that:

```
𝒟(V || W | P) < Δ                    (* (7) *)
```

Under this hypothesis, to prove our goal it is now sufficient to prove:

```
(minRate - cap) / 2 ≤ minRate - ℐ(P ; V)
```

Thanks to (7), we can prove `𝒟(V || W | P) ≤ (exp (-2))^2 / 2` and therefore use the lemma `mut_info_dist_ub` from the previous section (Sect. 12.3) to change `ℐ(P ; V)` to an expression using the capacity `cap`:

```
(minRate - cap) / 2 ≤ minRate -
  (cap + 1 / ln 2 * (|B| + |A| * |B|) * (- xlnx (sqrt (2 * 𝒟(V || W | P)))))
```

Using a few symbolic manipulations, we can show that to prove the latter goal it is actually sufficient to prove:

```
- xlnx (sqrt (2 * 𝒟(V || W | P))) ≤ gamma
```

Thanks to (6), to prove the latter goal it is sufficient to prove:

```
xlnx x ≤ xlnx (sqrt (2 * 𝒟(V || W | P)))
```

By (7), we know that `sqrt (2 * 𝒟(V || W | P)) < x`. The conclusion follows by the fact that `xlnx` is decreasing on the interval $[0, 1/e]$.

13 Formalization Overview

The complete formalization consists of 12,438 lines of Coq scripts; terseness owes much to the concise tactic language of SSReflect and its rich library. Table 1 provides a detailed account of the formalization in terms of script files. We have separated formalization of information theory from more mundane addenda to the Coq and SSReflect standard libraries. We rely in particular on SSReflect's canonical big operators, finite types, finite sets, and functions over finite domains, and on the standard Coq library for reals. To help navigation through the script files, we indicate the related sections in the article. The count in terms of lines of code distinguishes between specifications and proofs, and is provided by the standard `coqwc` command.

14 Related Work

The formalization of Shannon's theorems in this article as well as the formalization of advanced information-theoretic concepts (jointly typical sequences, channels, codes, the method of types, the error exponent bound, etc.) are new. As for the more basic concepts of information theory, one can find alternative formalizations in the literature. Based on the seminal work by Hurd [15], Coble formalizes (conditional) entropy and (conditional) mutual information [6], defines a notion of information leakage, and applies it to the verification of privacy properties of a protocol. Mhamdi et al. provides a formalization of the AEP and presents the source coding theorem as a potential application [17]; in other words, our work can be seen as the direct continuation of their work, though in a different proof-assistant.

For the purpose of this article, we formalized finite probability using SSReflect. As we have hinted at several times, this formalization was important to take advantage of SSReflect's library, and, in particular, of its canonical big operators [5]. For example, the genericity of SSReflect's canonical big operators was instrumental in the direct part of the channel coding theorem (Sect. 6.3) where we need sums over various kinds of objects (tuples, encoding functions, etc.) and with different operations (numerical addition, set union, etc.). We limit ourselves to finite probability because it is enough for our purpose (as for the information theory formalized by Coble [6]). Audebaud and Paulin-Mohring provide an alternative formalization of probabilities in Coq but that is biased towards verification of randomized algorithms [3]. Hasan et al. formalize probability theory on more general grounds in the HOL proof-assistant: they formalize the expectation properties [13] (this is also based on the work by Hurd [15]) and provide a formalization of the Chebyshev inequality and of the weak law of large numbers [16].

Our formalization of the source coding theorem follows Uyematsu [21, Chapter 1] with nevertheless much clarification (in particular, formalization of bounds). Our formalization of the channel coding theorem follows the detailed proofs of Hagiwara [12], with the difference that we got rid of extended reals in the formalization of the strong converse of the channel coding theorem. Part of our work was actually performed before the publishing of Hagiwara's textbook [12] and it was useful to unravel imprecisions, spot errors (regarding the usage of extended reals for example), and also discover a substantial hole: in the first version, the proof of the converse of the channel coding theorem was concluded by a short but

File	Reference in the article	spec	proof
Information Theory and Shannon's Theorems			
proba.v	Sect. 2	315	629
entropy.v	Sect. 3.1	22	47
aep.v	Sect. 3.1	42	103
typ_seq.v	Sect. 3.2	50	288
source_code.v	Sect. 4.1	24	0
source_coding_direct.v	Sect. 4.2	65	150
source_coding_converse.v	Sect. 4.3	62	194
channel.v	Sect. 5.1–5.2	118	112
binary_entropy_function.v	Sect. 5.3	18	110
binary_symmetric_channel.v	Sect. 5.3	46	174
joint_typ_seq.v	Sect. 5.4	65	306
channel_code.v	Sect. 6.1	30	18
channel_coding_direct.v	Sect. 6.2–6.3	107	754
num_occ.v	Sect. 8	148	429
types.v	Sect. 8	157	434
jtypes.v	Sect. 8	368	847
divergence.v	Sect. 9	27	85
conditional_divergence.v	Sect. 9	96	194
success_decode_bound.v	Sect. 10.1–10.2	142	189
channel_coding_converse.v	Sect. 11	48	113
log_sum.v	Sect. 12.1	15	183
variation_dist.v	Sect. 12.2	18	20
partition_inequality.v	Sect. 12.2.1	33	130
pinsker_function.v	Sect. 12.2.2–12.2.3	58	312
pinsker.v	Sect. 12.2.2–12.2.3	40	202
error_exponent.v	Sect. 12.3–12.4	52	189
	Total line count	2166	6212
Addenda to Standard Libraries			
Reals_ext.v	Addendum to the Coq library for reals	146	398
log2.v	Logarithm and exponential in base 2	49	169
Ranalysis_ext.v	Lemmas about derivability	62	192
ln_facts.v	Properties of functions with logarithm (e.g., xlnx used in Sect. 12.3–12.4)	84	524
ssr_ext.v	Addendum to the SSReflect library	151	530
tuple_prod.v	$(A \times B)^n \leftrightarrow A^n \times B^n$	45	63
Rssr.v	Coq reals with SSReflect-like naming	50	171
Rbigop.v	SSReflect bigops for Coq reals	196	677
arg_rmax.v	SSReflect argmax for Coq reals	81	85
natbin.v	Binary expressions of naturals (e.g., size_nat2bin used in Sect. 4.2)	107	280
	Total line count	971	3089

Table 1 Overview of the formalization of Shannon's theorems

spurious argument that we fixed with the lemma error_exponent_bound (Sect. 12.4) whose proof was known but turned out to be technical.

The perfect secrecy of one-time pad [20] is another famous theorem by Shannon. Its proof has been mechanized by Barthe et al. using EasyCrypt [4].

15 Conclusion and Future Work

We presented a formalization of the foundations of information theory in the SSReflect extension of the Coq proof-assistant. This formalization includes basic mathematical material such as finite probability, elementary information-theoretic

concepts such as typical sequences, but also more advanced notions such as jointly typical sequences, channels (illustrated with the example of the binary symmetric channel), codes (for source and channel coding), the basics of the method of types, information divergence and variation distance, all duly equipped with their standard properties.

We use this formalization to produce the first formal proofs of the two foundational theorems of information theory: the source coding theorem (direct part and converse part), that establishes the limit to reliable data compression, and the channel coding theorem (direct part and strong converse part), that establishes the limit to reliable data transmission over a noisy channel. Compared to pencil-and-paper proofs, our formalization has in particular the added value to make precise the construction of asymptotic bounds.

The channel coding theorem proves the existence of codes for reliable data transmission. Such codes play a critical role in IT products, e.g., LDPC (Low-Density Parity-Check) codes in storage devices. As a first step towards the verification of the implementation of error-correcting codes, we have been working on formalizing their basic properties. At the time of this writing, we have already extended the work presented in this article with formalizations of basic properties of linear, cyclic and Hamming codes (see [2]), and we are now tackling the formalization of LDPC codes.

Acknowledgements The first author acknowledges the help of several members of the development team of SSReflect for the usage of the SSReflect library. The authors are grateful to the anonymous reviewers whose valuable comments significantly improved this article.

References

1. Affeldt, R., Hagiwara, M.: Formalization of Shannon's Theorems in SSReflect-Coq. In: Proceedings of the 3rd International Conference on Interactive Theorem Proving (ITP 2012), Princeton, NJ, USA, August 13–15, 2012. Lecture Notes in Computer Science, vol. 7406, pp. 233–249. Springer, Heidelberg (2012)
2. Affeldt, R., Hagiwara, M., Sénizergues, J.: Formalization of Shannon's Theorems in SSReflect-Coq. Coq documentation and scripts available at `http://staff.aist.go.jp/reynald.affeldt/shannon`.
3. Audebaud, P., Paulin-Mohring, C.: Proofs of randomized algorithms in Coq. Sci. Comput. Program. 74(8):568–589 (2009)
4. Barthe, G., Crespo, J. M., Grégoire, B., Kunz, C., Zanella Béguelin, S.: Computer-Aided Cryptographic Proofs. In: Proceedings of the 3rd International Conference on Interactive Theorem Proving (ITP 2012), Princeton, NJ, USA, August 2012. Lecture Notes in Computer Science, vol. 7406, pp. 11–27. Springer, Heidelberg (2012)
5. Bertot, Y., Gonthier, G., Ould Biha, S., Pasca, I.: Canonical Big Operators. In: Proceedings of the 21st International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2008), Montreal, Canada, August 18–21, 2008. Lecture Notes in Computer Science, vol. 5170, pp. 86–101. Springer, Heidelberg (2008)
6. Coble, A.R.: Anonymity, Information, and Machine-Assisted Proof. PhD Thesis, King's College, University of Cambridge, UK (2010)
7. The Coq Development Team. Reference Manual. Version 8.4. Available at `http://coq.inria.fr`. INRIA (2004-2012)
8. Cover, T.M., Thomas, J.A.: Elements of Information Theory, 2nd edition. Wiley-Interscience (2006)
9. Csiszár, I.: The Method of Types. IEEE Transactions on Information Theory 44(6), 2505–2523 (1998)
10. Csiszár, I., Körner, J.: Information Theory—Coding Theorems for Discrete Memoryless Systems. Second edition. Cambridge University Press (2011)

11. Gonthier, G., Mahboubi, A., Tassi, E.: A Small Scale Reflection Extension for the Coq system. Version 10. Technical report RR-6455. INRIA (2011)
12. Hagiwara, M.: Coding Theory: Mathematics for Digital Communication. In Japanese. http://www.nippyo.co.jp/book/5977.html. Nippon Hyoron Sha, 2012
13. Hasan, O., Tahar, S.: Verification of Expectation Using Theorem Proving to Verify Expectation and Variance for Discrete Random Variables. J. Autom. Reasoning 41:295–323 (2008)
14. Hölzl, J., Heller, A.: Three Chapters of Measure Theory in Isabelle/HOL. In: Proceedings of the 2nd International Conference on Interactive Theorem Proving (ITP 2011), Berg en Dal, The Netherlands, August 22–25, 2011. Lecture Notes in Computer Science, vol. 6898, pp. 135–151. Springer, Heidelberg (2011)
15. Hurd, J.: Formal Verification of Probabilistic Algorithms. PhD Thesis, Trinity College, University of Cambridge, UK (2001)
16. Mhamdi, T., Hasan, O., Tahar, S.: On the Formalization of the Lebesgue Integration Theory in HOL. In: Proceedings of the 1st International Conference on Interactive Theorem Proving (ITP 2010), Edinburgh, UK, July 11–14, 2010. Lecture Notes in Computer Science, vol. 6172, pp. 387–402. Springer, Heidelberg (2010)
17. Mhamdi, T., Hasan, O., Tahar, S.: Formalization of Entropy Measures in HOL. In: Proceedings of the 2nd International Conference on Interactive Theorem Proving (ITP 2011), Berg en Dal, The Netherlands, August 22–25, 2011. Lecture Notes in Computer Science, vol. 6898, pp. 233–248. Springer, Heidelberg (2011)
18. Khudanpur, S.: Information Theoretic Methods in Statistics. Lecture notes (1999). Available at http://old-site.clsp.jhu.edu/~sanjeev/520.674/notes/GISAlgorithm.pdf (last access: 05/02/2013)
19. Shannon, C.E.: A Mathematical Theory of Communication. Bell System Technical Journal, vol. 27, pp. 379–423 and 623–656. 1948
20. Shannon, C.E.: Communication Theory of Secrecy Systems. Bell System Technical Journal, vol. 28, pp. 656–715. 1949.
21. Uyematsu, T.: Modern Shannon Theory, Information Theory with Types. In Japanese. Baifukan (1998)
22. Verdú, S.: Fifty Years of Shannon Theory. IEEE Transactions on Information Theory 44(6):2057–2078 (1998)