Examples of Formal Proofs about Data Compression

Reynald Affeldt  
National Institute of Advanced Industrial Science and Technology

Jacques Garrigue  
Grad. School of Mathematics Nagoya University

Takafumi Saikawa  
Grad. School of Mathematics Nagoya University

Abstract— Because of the increasing complexity of mathematical proofs, there is a growing interest in formalization using proof-assistants. In this paper, we explain new formal proofs of standard lemmas in data compression (Jensen’s and Kraft’s inequalities) as well as concrete applications (to the analysis of compression methods and Shannon-Fano codes). We explain in particular how one turns the paper proof into formal terms and the relation between the informal proof and the formal one. These formalizations come as an extension to an existing formal library for information theory and error-correcting codes.

I. MOTIVATION AND CONTENTS

A proof-assistant is a piece of software to check mathematical proofs written in formal logic. A formal logic is essentially a minimum set of non-contradictory axioms. Most proof-assistants use (variants of) type theory, whose expressiveness is equivalent to set theory. In a proof-assistant, the only trusted type theory assistants use (variants of) a minimum set of non-contradictory axioms. Most proof-cal proofs written in formal logic. A formal logic is essentially for information theory and error-correcting codes.

In the past years, we have been working on the formalization of information theory and error-correcting codes for the COQ proof-assistant [10] using the MATHTCOMP library [11]. The result is another library (called INFOTHEO [13]) that made it possible to verify Shannon’s theorems [4], [5], as well as error-correcting codes [6], [8].

Our goal is now to turn the INFOTHEO library into a practical tool. To achieve this goal, we are now involved in the following activities: enrich the library with new lemmas, make it easier to use already-formalized results to prove new ones, and publicize the whole library to potential users. These are the three goals of this paper. Concretely, in this paper, we provide new formal proofs of standard lemmas in data compression (namely, Jensen’s and Kraft’s inequalities), work out concrete applications (analysis of compression methods and Shannon-Fano codes), and explain our results so as to be understood by readers who may not be familiar with proof-assistants.

The only effort that we require from our readers is to cope with the syntax of formal logic (as implemented by the COQ proof-assistant). It is not difficult to decipher formal statements, especially when one already has the corresponding mathematical background. Indeed, modern parsing technologies allow for familiar \texttt{let}\texttt{in}\texttt{X} like notations. In contrast, we do not expect the reader to read the details of proof scripts (the sequence of commands that symbolically manipulate formal statements to prove a goal) but we display them to show how (well) they match paper proofs.

II. JENSEN’S INEQUALITY AND ANALYSIS OF COMPRESSION METHODS

A. Jensen’s Inequality, Informally

Jensen’s inequality is essentially a generalization of concavity for a finite set of points. Concavity of a real function $f$ can be stated as:

$$\forall a, b > 0. \quad \frac{af(x) + bf(y)}{a + b} \leq f\left(\frac{ax + by}{a + b}\right).$$

Jensen’s inequality is a consequence of concavity, and can be stated as:

$$\forall a_1, \ldots, a_n > 0. \quad \frac{\sum a_i f(x_i)}{\sum a_i} \leq f\left(\frac{\sum a_i x_i}{\sum a_i}\right).$$

B. Jensen’s Inequality, Formally

While the above definition gives the gist of Jensen’s inequality, when formalizing it is important both to use simple definitions (with few variables and conditions), and to make sure that the proven theorem can be applied in a wide range of situations. As we want it to apply to locally concave functions, we start by defining intervals, as convex subsets of $\mathbb{R}$.

Definition convex_interval (D:R -> Prop) := forall x y t, D x -> D y -> 0 <= t <= 1 -> D (t * x + (1-t) * y).

Record interval := mkInterval { mem_interval : R -> Prop; interval_convex : convex_interval mem_interval }.

We then define concavity, using intervals as predicates.

Definition concave_leq (f : R -> R) (x y t : R) := f (t * x + (1-t) * y) <= f t * x + (1 - t) * y.

Definition concave_in (D : interval) (f : R -> R) := forall x y t : R, D x -> D y -> 0 <= t <= 1 -> concave_leq f x y t.

When stating Jensen’s inequality, we replace the points and weights by a function $r$ and a distribution $X$, both defined over a finite domain $A$.

Variables (f : R -> R) (D : interval).

Hypothesis concave_f : concave_in D f.

Variable A : finType.

Theorem jensen_dist_concave (r : A -> R) (X : dist A) : (forall x, D (r x)) -> \sum_(a in A) f (r a) * X a <= f (\sum_(a in A) r a * X a).

Using a distribution tells us that the weights are non-negative, and their sum is 1, in the same way as interval brings its
invariant. This theorem is proved by induction on the size of the support of the distribution (i.e., the α's of a such that \( x \alpha \neq 0 \)) [2, Thm. 2.6.2]. Formally, we follow this approach by first defining such an induction principle, and using it to structure the proof, which as a result is only about 60 lines long. Note that the original proof used a binary distribution as base case, but it is simpler and more general to use a trivial unary distribution (i.e., \( x a = 1 \) for a single \( a \in A \)).

C. Application: Analysis of Compression Methods

A simple application of Shannon’s inequality is comparing the zero-order entropy of strings to that of their concatenation.

Let \( N^S_s \) be the number of occurrences of the letter \( s \) in the string \( S \). The zero-order empirical entropy of a string \( S \) of length \( n = |S| \) over the alphabet \( \Sigma = \{s_1, \ldots, s_\sigma\} \) is defined using the Shannon entropy of its observed probabilities [9, Sec. 2.3.2]:

\[
H_0(S) = \mathcal{H} \left( \frac{N^S_{s_1}}{n}, \ldots, \frac{N^S_{s_\sigma}}{n} \right) = \sum_{s \in \Sigma} \frac{N^S_s}{n} \log \frac{n}{N^S_s}.
\]

This leads to the simple formula: \( nH_0(S) = \sum_{s \in \Sigma} N^S_s \log \frac{n}{N^S_s} \).

If we consider two strings \( S_1 \) and \( S_2 \), of lengths \( n_1 \) and \( n_2 \) and their concatenation \( S \) of length \( n = n_1 + n_2 \), we have

\[
n_1H_0(S_1) + n_2H_0(S_2) = \sum_{s \in \Sigma} N^S_{S_1} \log \frac{n_1}{N^S_{S_1}} + \sum_{s \in \Sigma} N^S_{S_2} \log \frac{n_2}{N^S_{S_2}}
\leq \sum_{s \in \Sigma} (N^S_{S_1} + N^S_{S_2}) \log \frac{n_1 + n_2}{N^S_{S_1} + N^S_{S_2}} = nH_0(S),
\]

where the inequality is obtained by applying Jensen at each \( s \) with parameters \( a_1 = N^S_{S_1}, x_1 = \frac{n_1}{n}, f = \log \) [9, Sec. 2.8].

To make the example more convincing, we will apply it to the concatenation \( S \) of \( \ell \) strings \( S_i \) (of length \( n_i \)). Namely,

\[
\sum_{i=1}^{\ell} n_iH_0(S_i) \leq nH_0(S), \text{ or in Coq's language:}
\]

```coq
Variable A : finType.
Definition nhs (s : seq A) :=
  \sum_{a \in A} if N(a|s) == 0%nat then 0 else N(a|s) * log (size s / N(a|s)).

Theorem concats_entropy (ss : seq (seq A)) :
  \forall (s <- ss) nhs s <= nhs (flatten ss).
```

Here, \( \text{nhs} \) is the formalization of the function \( |S|H_0(S) \).

We represent strings as sequences of symbols from an alphabet \( A \). The length of a sequence \( s \) is size \( s \). \( N(a|s) \) is a notation for \( N^S_s \) \( \\text{\sum}_{a \in A} \) stands for the iterated sum\(^1 \) \( \sum_{a \in A} \).

In the formal statement, we represent the \( \ell \) strings \( S_i \) as a sequence of sequences \( ss \), so that we sum directly over \( ss \) instead of the indices \( \{1, \ldots, \ell\} \), and concatenate the strings using the \text{flatten} operation on this sequence.

Like we saw in the informal definition above, \( \text{nhs} \) is related to the entropy of the symbol distribution as follows:

```coq
Lemma szhhs_is_nhs (s : seq A) (H : size s != 0) :
  size s * H (\text{\sum}_{a \in A} \text{occ_dist} s H) = nhs s.
```

\(^1\)Note that definitions may contain implicit coercions: size or \( N(a|s) \) are natural numbers, but the arithmetic operators expect real numbers, so that a coercion from \( \mathbb{N} \) to \( \mathbb{R} \) is added. It can also be written explicitly as \( \text{\text{nat} R} \).

The formal proof of \( \text{concat\_entropy} \) is shown in Fig. 1. Rather than just seeing it as an artefact, we will try to show how close it is to a paper proof. Some of the theorems used for transformations are given in Fig. 2. The proof goes through the following steps, which are marked in the script.

1) Expand definitions and inverse the order of the summations (\text{exchange\_big}), so that the goal becomes:

\[
\sum_{s \in \Sigma} \sum_{i=1}^{\ell} \text{if } N^S_{S_i} = 0 \text{ then } 0 \text{ else } N^S_{S_i} \log \frac{|S|}{N^S_{S_i}} \leq \sum_{s \in \Sigma} \sum_{i=1}^{\ell} \text{if } N^S_{S_i} = 0 \text{ then } 0 \text{ else } N^S_{S_i} \log \frac{|S|}{N^S_{S_i}}.
\]

2) Show that it is sufficient to prove the inequalities for each \( s \), removing strings which contain no occurrences of \( s \) (for simplicity of the indexing, let us pretend that the \( \ell' \) first strings contain occurrences, the formal proof does a reordering).

\[
\sum_{i=1}^{\ell'} N^S_{S_i} \log \frac{|S|}{N^S_{S_i}} \leq \sum_{i=1}^{\ell'} \log \frac{|S|}{\sum_{i=1}^{\ell} N^S_{S_i}}
\]

Note that \( |S| \) on the right-hand side still contains the lengths of the omitted strings. This proof makes repeated use of \text{big\_ID} and \text{big\_1} to separate and remove those strings in other sums.

3) Prove the inequality

\[
\left( \sum_{i=1}^{\ell'} N^S_{S_i} \right) \log \frac{\sum_{i=1}^{\ell'} |S_i|}{\sum_{i=1}^{\ell} N^S_{S_i}} \leq \left( \sum_{i=1}^{\ell'} N^S_{S_i} \right) \log \frac{\sum_{i=1}^{\ell'} |S_i|}{\sum_{i=1}^{\ell} N^S_{S_i}}
\]

using the monotonicity of \( \log \). Here the reasoning mostly involves real arithmetic, with the need to prove many side conditions on lemmas. In the formal proof, to conclude we need to work with the concrete indexing, which uses a filtered sequence rather than \( \ell' \).

4) Define the distribution \( d \) and the point function \( r \) as:

\[
d(i) = \frac{N^S_{S_i}}{\sum_{i=1}^{\ell'} N^S_{S_i}} \quad \text{r}(i) = \frac{|S_i|}{N^S_{S_i}}
\]

and prove that \( r(i) \) is strictly positive. In the script, \text{seq\_nat\_dist} builds a distribution from a non-zero summation of natural numbers, \text{in\_tuple} turns a sequence into a tuple, which is just a fixed-length sequence, and \text{nth} extracts its \( i \)th element.

5) Use these proofs and the concavity of \( \log \) to apply \text{jensen\_dist\_concave}. This gives:

\[
\sum_{i=1}^{\ell'} \left( \log \frac{|S_i|}{N^S_{S_i}} \right) \frac{N^S_{S_i}}{\sum_{i=1}^{\ell'} |S_i|} \leq \log \left( \sum_{i=1}^{\ell'} \frac{|S_i|}{N^S_{S_i}} \sum_{i=1}^{\ell'} \frac{N^S_{S_i}}{\sum_{i=1}^{\ell'} N^S_{S_i}} \right).
\]

The formal proof uses \text{big\_nth} to convert from summing using numerical indices, as required by \text{jensen\_dist\_concave}, to summing directly on the elements of the sequence, which is the preferred approach throughout this proof. We choose not to distinguish both versions in the mathematical notation.

6) Multiply both sides by \( \sum_{i=1}^{\ell'} N^S_{S_i} \), and use distribution laws and algebraic laws to obtain:
Theorem concats_entropy ss : 
\sum_{i < ss} N(a|s) \leq \log \left( \frac{\sum_{i < ss} N(a|s)}{N(a|\text{flatten ss})} \right)
\leq \log \left( \frac{N(a|\text{flatten ss})}{N(a|\text{flatten ss})} \right).

Proof.

apply leq_addl by rewrite /leq_addl /leqP /count_size.

Qed.

(* (3) Prepare to use jensen_dist_concave *)

apply (@leq_trans N(a|tnth (in_tuple ss _ _ i) _ _ i)).

apply big_tnth Hnum2; apply /big_addn /bigID.

(* (4) Prepare to use jensen_dist_concave *)

have Htotl : emsy (num_occ_flatten a ss').

rewrite big_tnth in Htotal.

have Hnum2 : N(a|flatten ss') = 0.

rewrite -lt0n Hsum.

apply /leq_addl; apply leqP.

(* (5) Apply Jensen *)

move (lensen_dist_concave d Hr).

rewrite /d /r /=.

move/Rle_trans; apply. (* LHS matches *)

(* (5) Apply Jensen *)

move (lensen_dist_concave d Hr).

rewrite -lt0n Hsum.

apply /leq_addl; apply leqP.

(* (6) Transform the statement to match the goal *)

Premise is as in (5), goal as in (6)

move (lensen_dist_concave d Hr).

apply big_tenth1 /

rewite (eqP Hsum) /eq_bigr /bigID.

rewrite (eq_bigr Hnum).
B. Kraft’s Inequality, Informally

Kraft’s inequality is a necessary and sufficient condition for the existence of a prefix code. It says that, given the lengths \( \ell_0, \ldots, \ell_{n-1} \), there exists a prefix code with \( n \) codewords \( C_i \) s.t. \( |C_i| = \ell_i \) iff \( \sum_{i<n} \ell_i \leq 1 \), where \( T \) is the alphabet for the codewords.

To prove the direct part, let \( \ell_{\max} \) be the largest length. Then:

\[
\sum_{i<n} |C_i|^{\ell_{\max} - \ell_i} = \sum_{i<n} \{ x \mid \text{prefix } C_i, x \} \tag{III.1}
\]

\[
= \bigcup_{i<n} \{ x \mid \text{prefix } C_i, x \} \tag{III.2}
\]

\[
\leq |T|^{\ell_{\max}}. \tag{III.3}
\]

In particular, the equality (III.1)–(III.2) holds thanks to the prefix property. See Sec. III-C2 for a complete formal proof.

3) Kraft’s Inequality (reverse part): We now complete the informal proof of the converse of Kraft’s inequality (started in Sec. III-B) and provide a complete proof script. We were left with two subgoals, \( r = \frac{w_k}{|T|^{\ell_k}} \geq w_j + 1 \) is proved as follows:

\[
r = \sum_{i<k} |T|^{\ell_i} - |T|^{\ell_j} = w_j + \sum_{j<k} |T|^{\ell_i} - |T|^{\ell_j} \geq w_j + 1. \tag{III.4}
\]

Here follows the proof for \( r - 1 < w_j \):

\[
r - 1 = \frac{w_k}{|T|^{\ell_k}} - 1 \quad \text{(by definition) \quad (III.5)}
\]

\[
= w_j + \sum_{j<k} |T|^{\ell_i} - |T|^{\ell_j} - 1 \quad \text{(III.6)}
\]

\[
< w_j. \tag{III.9}
\]

The fact that \( \sigma_j \) is a prefix of \( \sigma_k \) is used in the step (III.7)–(III.8). These two subgoals respectively correspond to the subgoals have H1 and have H2 below:

**Lemma kraft_implies_prefix :** kraft_code R T \( \vdash \) exists C : code_set T, prefix_code C.

**Proof.**

\[
\text{move} \vdash \exists (\text{ACode } _l\text{n sorted}_l). \quad \text{apply } \text{nmap_prefix}. \\
\text{at this point, the goal is } w_j, k < k \rightarrow \text{prefix } \sigma_j \sigma_k.
\]

**Lemma prefix_implies_kraft :** prefix_code C \( \vdash \) exists \( |T|^{\ell_k} - |T|^{\ell_j} \geq 1 \).

**Proof.**

\[\vdash \exists (\text{ACode } _l\text{n sorted}_l). \quad \text{apply } \text{nmap_prefix}. \\
\text{at this point, the goal is } w_j, k < k \rightarrow \text{prefix } \sigma_j \sigma_k.
\]
D. Application: Shannon-Fano Codes

Let us assume a source that emits symbols from an alphabet \( A \) with probability \( P_r \). A Shannon-Fano code is such that each symbol \( a \) is mapped to a codeword of length \( \left\lceil \log \frac{1}{P(a)} \right\rceil \).

This definition requires to make explicit the encoding function from \( A \) to the codewords as a type encoding \( t \):

\[
\text{Record} \; t \; (A \; T \; : \; \text{FinType}) := mk \{ \text{f} \; : \; (\text{ffun} \; A \rightarrow \; \text{seq} \; T); \text{f_inj} \; : \; \text{injective} \; \text{f} \; \}. 
\]

Let \( P \) be a distribution over \( A \). Formally, a Shannon-Fano code is an encoding function that satisfies the following predicate:

\[
\text{Definition} \; \text{is_shannon_fano} \; (f \; : \; \text{Encoding} \; t \; A \; T) := \\
\quad \forall a, \; \text{size} \; (f \; a) = \\
\quad Zabsnat \; (\text{ceil} \; (\text{Log} \; \#|T| \times \%R \; (1 / P \; a))). 
\]

(Zabsnat takes the absolute value of an integer.)

We can show that the list of codewords generated by a Shannon-Fano code satisfies the Kraft condition:

\[
\text{Definition} \; \text{is_shannon_fano \_is_kraft} : \\
\quad \text{is_shannon_fano} \; \mathcal{F} \rightarrow \text{kraft\_condR} \; T \; \text{sizes}.
\]

In consequence, the construction of Sec. III-A provides a way to construct a (prefix) Shannon-Fano code.

For the sake of completeness, we establish formally the fact that Shannon-Fano codes are sub-optimal. It suffices to show that the average code length is less than \( H(P) + 1 \):

\[
\text{Lemma} \; \text{shannon_fano \_suboptimal} : \\
\quad \text{is_shannon_fano} \; \mathcal{F} \rightarrow \; \text{average} \; \mathcal{F} \; < \; 1 \; \times \; H \; \mathcal{P} \; + \; 1.
\]

IV. RELATED WORK

There already exist several examples of formal proofs about data compression. Shannon’s source coding theorems have been formalized in the COQ proof-assistant [4], [5], the algorithmic aspects (but not the information-theoretic aspects) of Huffman codes have been formalized in Isabelle/HOL [3]. These examples are larger than the ones discussed in this paper but our work still improves on previous work: our proof of Kraft’s inequality covers alphabets of any size (the proof of Kraft’s inequality in [7] is for the binary case), our proof of Jensen’s inequality covers partial functions (the proof in [12] is for total functions). One could argue that these are minor improvements but they are moreover integrated within the same library for formalization of information theory and error-correcting codes [13], allowing for further applications.

V. CONCLUSION AND FUTURE WORK

We now provide new formalizations about data compression (Jensen’s and Kraft’s inequalities) as well as two concrete applications (analysis of string compression and Shannon-Fano codes). We tailored our technical explanations for a reader who is not proficient neither in formal logic nor with proof-assistants, by explaining in particular how one gets from the paper proof to a formal proof. Our hope is to improve the usability of our library for formalization of information theory and error-correcting codes.

Proof scripts are available online [13]. The files relevant to this paper are: jensen.v (Sec. II-B), string_entropy.v (Sec. II-C), kraft.v (Sec. III-C), and shannon_fano.v (Sec. III-D). There were also several technical improvements to the INFOTHEO library (mostly about the interface with Coq’s standard library for real analysis).

We plan a number of technical improvements: support for a version of the MATHCOMP library with well-integrated real numbers, better formal definitions of codes and encoding functions. (We will then generalize results (for example from Kraft’s inequality to Kraft-McMillan’s) to tackle other codes. The resulting formal theory of data compression should be large enough to help us verify software implementations of compact data structures [9, Chap. 2].

REFERENCES