About the Formal Verification of Shannon's Theorems

Reynald Affeldt
Research Institute for Secure Systems
National Institute of Advanced Industrial Science and Technology

The proof of the four color theorem by Appel and Haken in 1976 [3] is a good example of the usefulness of computers in the conception of mathematical proofs. At first, some mathematicians disregarded this proof because it relied on an enumeration by a computer program that cannot be checked by hand. Yet, there are more recent examples of proofs, such as the proof of the Kepler conjecture by Hales [7], that are anyway considered uncheckable with 100% certainty by even the most qualified experts. Computer-driven interactive proofs recently emerged as the solution to guarantee the correctness of such large mathematical proofs.

Software for interactive proofs of mathematics take the form of computer languages for the management and the writing of mathematical facts and their proofs. It is important for such software to keep their trusted base as small as possible. This is the design principle behind proof-assistants such as Coq [4]. The kernel of Coq is an implementation of *type theory*, an alternative to set theory for the foundations of mathematics [8]. In this setting, mathematical facts are expressed using types `T` and their proofs `P` are written as functional programs, with the dedicated notation `P : T`. This analogy with programming languages has been known since 1968 as the *Curry-Howard correspondence*. Since there is an algorithm to check that a program matches its type, Coq is able to provide automatic checking of proofs, relieving humans of this burden.

Compared with work cited above, our purpose here is very modest. We just want to illustrate formal verification using an example from coding theory. We will provide a detailed explanation of the formalization of Shannon's source coding theorem (a.k.a. the noiseless coding theorem) using SSReflect [5], an extension to Coq that provides a library that was used in particular to verify the Feit-Thompson theorem [6]. The present document is actually an extended version of the first half of [1]. For details about the formalization of the direct part of the noisy channel coding theorem see [1], for the formalization of the converse of the noisy channel coding theorem see [2], for the matching pencil-and-paper presentation see [9].

tag in script

We display the formalization as it is processed by Coq. (Between `/*` and `*/` are comments; ... are for elided parts.) Margin comments are tags referring to the proof script. The lecture material is available at [2]. Interactive proof will be demonstrated during the lecture. Various Coq tutorials are available online [4].

1 Finite Probabilities

1.1 Distributions

A probability space will be the powerset of some finite type `A` (type `finType` in SSReflect). An *event* is a subset of the probability space and can therefore be encoded as a boolean predicate, of type `pred A` in SSReflect. An elementary event is an event that cannot be decomposed further, i.e., a predicate `pred1 a` ≈ `fun x ⇒x = a` (for some `a` ∈ `A`). A *distribution* associates a positive real to each elementary event s.t. the sum for all elementary events is 1. The probability of an event is the sum of the probabilities of its elementary events.

1 **Definition 1** (Distribution)**.** Given a finite type `A`, a distribution is defined as a positive real-valued (probability mass) function `pmf`, equipped with a proof `pmf1` that its outputs sum to 1:

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

A distribution is thus a `Record` but thanks to the coercion `:>`, we can write "`P a`" as a function application to represent the probability associated with `a`. The construct `Σ_(a in A) F a` is provided by SSReflect's bigops theory, that is further illustrated by the next two examples.

2 **Example 1** (Product Distribution)**.** Let us assume that we are given two distributions `P₁: dist A` and `P₂: dist B`. The *product distribution* over `A×B` noted `P₁×P₂` is defined as follows:

```
Definition prod_dist : dist [finType of A * B].
apply mkDist with (fun ab ⇒ P₁ ab.1 * P₂ ab.2).
- /* Rle0f proof */ ...
- /* pmf1 proof */ /* goal: Σ_(a|a ∈ [finType of A * B]) P₁ a.1 * P₂ a.2 = 1 */
  rewrite -(pair_big xpredT xpredT (fun a b ⇒ P₁ a * P₂ b)) /=.
  /* Σ_(a|a ∈ A) Σ_(b|b ∈ B) P₁ a * P₂ b = 1 */
  rewrite -(pmf1 P₁).
  /* Σ_(a|a ∈ A) Σ_(b|b ∈ B) P₁ a * P₂ b = Σ_(a|a ∈ A) P₁ a */
  apply eq_bigr ⇒ a _.
  /* Σ_(b|b ∈ B) P₁ a * P₂ b = P₁ a */
  rewrite -big_distrr /=.
  /* P₁ a * (Σ_(b|b ∈ B) P₂ b) = P₁ a */
  rewrite pmf1.
  /* P₁ a * 1 = P₁ a */
  ...
Defined.
```

3 **Example 2** (Tuple Distribution)**.** Let us assume that we are given a distribution `P : dist A`. The *tuple distribution* over A^n noted `P^n` is defined as follows:

```
Definition tuple_dist : dist [finType of n.-tuple A].
apply mkDist with (fun x ⇒ Π_(i < n) P x_i).
- /* Rle0f proof */ ...
- /* pmf1 proof */ /* goal: Σ_(t|t ∈ [finType of n.-tuple A]) Π_(i<n) P t_i = 1 */
  pose P' := fun (a : 'I_n) b ⇒ P b.
  suff : Σ_(f : {ffun 'I_n → A }) Π_(i < n) P' i (f i) = 1.
```

```
    ...
  /* Σ_(f : {ffun 'I_n → A }) Π_(i < n) P' i (f i) = 1 */
  rewrite -bigA_distr_bigA /= /P'.
  /* Π_(i<n) Σ_(a in A) P a = 1 */
  rewrite [X in _ = X](_ : 1 = Π_(i < n) 1); last by ...
  /* Π_(i<n) Σ_(j in A) P j = Π_(_<n) 1 */
  apply eq_bigr ⇒ i _.
  /* Σ_(j in A) P j = 1 */
  ...
Defined.
```

`n.-tuple A` is SSReflect notation for A^n. Given a tuple `t` ∈ `n.-tuple A`, `t_i` is notation for the `i`th component of `t`, with `i` ∈ `'I_n` is a natural strictly smaller than `n` (an *ordinal* in SSReflect's parlance).

4 **Definition 2** (Probability of an Event)**.** Given a distribution `P : dist A` and an event `E : pred A`, the probability of `E` (w.r.t. distribution `P`) is defined as follows:

```
Definition Pr (E : pred A) := Σ_(a in A | E a) P a.
```

5 **Lemma 1.** Some properties of the probability of events for the sake of completeness (they will be used later). (`[predC E]` is the complement of predicate `E`.)

```
Lemma Pr_incl (E E' : pred A) : (∀ i, E i → E' i) → Pr E ≤ Pr E'.
Lemma Pr_cplt E : Pr E + Pr [predC E] = 1.
```

1.2 Random Variables

The restriction to finite probabilities makes it possible to simplify the definition of random variables.

6 **Definition 3** (Random Variable)**.** A *random variable* (RV) is a distribution over some finite type `A` coupled with a real-valued function defined over `A`:

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

7 The (implicit) probability space is the powerset of `A`. We note *p*_`X` the distribution of the RV `X`. We note `Pr[X = r]` the following definition :

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

8 (`[pred x | .. x ..]` is a SSReflect notation for boolean predicates.) We note `X` − `m` the RV that subtracts `m` to the output of `X`, `k` × `X` the RV that multiplies by `k` the output of `X`, `X` / `n` the RV that divides by `n` the output of `X`, `X`ˆ2 the RV that squares the output of `X`, − log `P` the RV `fun x ⇒- log(P x)` (with underlying distribution `P`).

9 **Definition 4** (Expected Value)**.** The *expected value* of a RV `X` (over `A`) is noted $\mathcal{E}$ `X` and is defined as follows:

```
Definition Ex_alt := Σ_(a in A) X a * p_X a.
```

10 This is equivalent to the following definition:

```
Definition img (A : finType) (f : A → R) := undup (map f (enum A)).
Definition Ex := Σ_(r ← img X) r * Pr[ X = r ].
```

The expected value is to be understood as the average outcome; this will become clearer when we state the weak law of large number (Lemma 5).

The *variance* is a measure of the deviation of the outcome of an experiment from its expected value.

11 **Definition 5.** Given a RV `X`, the *variance* of `X` is noted $\mathcal{V}$ `X` and is defined as follows:

```
Definition Var := let μ := ℰ X in ℰ ((X − μ)^2).
```

12 **Lemma 2.** The variance can also be computed as follows:

```
Lemma V_alt : Var = ℰ (X ^2)  - (ℰ X)^2.
```

The Chebyshev Inequality bounds the probability of deviations from the expected value.

13 **Lemma 3** (Chebyshev Inequality)**.** Given a RV `X` over `A`, the following holds:

```
Lemma chebyshev_inequality ε : 0 < ε →
  Pr p_X [pred a | Rabs (X a - ℰ X) ≥ ε] ≤ 𝒱 X / ε ^ 2.
```

Proof. Script excerpt:

```
...
/* ε ^ 2 * Pr p_X [pred a | Rabs (X a - ℰ X) ≥ ε] ≤ 𝒱 X */
rewrite /𝒱 {2}/Ex_alt.
/* lhs ≤ Σ_(a|a ∈ A) ((X − ℰ X) ^2) a * p_((X − ℰ X) ^2) a */
rewrite (_ : p_((X − ℰ X)^2) = p_X) //.
/* lhs ≤ Σ_(a|a ∈ A) ((X − ℰ X) ^2) a * p_X a */
apply Rle_trans with (Σ_(a in A | Rabs (X a - ℰ X) ≥ ε) (((X − ℰ X)^2) a
* p_X a));
  last by ...
/* ε ^ 2 * Pr p_X [pred a | Rabs (X a - ℰ X) ≥ ε] ≤
   Σ_(a|(a ∈ A) ∧ (Rabs (X a - ℰ X) ≥ ε)) ((X − ℰ X) ^2) a * p_X a */
rewrite /Pr big_distrr [_ ^2]lock /= -!lock.
/* Σ_(a|Rabs (X a - ℰ X) ≥ ε) ε ^ 2 * p_X a ≤
   Σ_(a|Rabs (X a - ℰ X) ≥ ε) ((X − ℰ X) ^2) a * p_X a */
...
```

□

1.3 Sum of Random Variables

14 **Definition 6** (Joint Distribution)**.** The fact that `P` is a *joint distribution* of P_1 (distribution over `A`) and P_2 (distribution over A^n) is captured by the following predicate:

```
(∀ 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} | behead t  =  x) P t).
```

(`thead` takes the head of a tuple, `behead` its tail.) So, when a distribution `P` is a joint distribution of two distributions P_1 and P_2, P_1 and P_2 are two marginal distributions of `P`. Still, a joint distribution is not necessarily a product distribution.

15 **Definition 7** (Independent Random Variables)**.** The fact that a RV `X` (over `A`) and a RV `Y` (over A^n) are independent w.r.t. the joint distribution `P` (over A^{n+1}) is noted $X \perp_P Y$ and holds when:

```
∀ x y, Pr P [pred xy | (X (thead xy)  =  x) ∧ (Y [tuple of behead xy]  =  y)] =
       Pr[X = x] * Pr[Y = y].
```

16 **Definition 8** (Sum of Two Random Variables)**.** The fact that `X` (RV over A^{n+2}) is the sum of X_1 (RV over `A`) and X_2 (RV over A^{n+1}) is noted $X = X_1 + X_2$ and is formalized as follows:

```
Definition sum := joint p_X_1 p_X_2 p_X ∧
  X =_1 [ffun x ⇒ X_1 (thead x) + X_2 [tuple of behead x]].
```

17 **Lemma 4** (Expected Value and Variance of Sum of RVs)**.** The expected value is always linear but the variance only when the RVs are independent:

```
Lemma E_linear_2 : X = X_1 + X_2 → ℰ X = ℰ X_1 + ℰ X_2.
```

18

```
Lemma V_linear_2 : X = X_1 + X_2 → X_1  ⊥_p_X  X_2  → 𝒱 X = 𝒱 X_1 + 𝒱 X_2.
```

19 **Definition 9** (Sum of Independent Random Variables)**.** We are given a RV `X` over A^n and a n-tuple `Xs` of random variables (each over `A`). The fact that `X` is the sum of the RVs in `Xs` considered as independent is noted $X =_i \Sigma\, Xs$ and is captured by the following predicate:

```
Inductive isum_n : ∀ n, rvar [finType of n.-tuple A] → n.-tuple (rvar A) → Prop :=
| isum_n_1 : ∀ X, cast_rv X =_i Σ X
| isum_n_cons : ∀ n (Ys : n+1.-tuple _) Y X Z,
  Y =_i Σ Ys  →  Z = X + Y → X  ⊥_p_Z  Y  →  Z =_i Σ [tuple of X :: Ys]
```

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 (thus justifying the interpretation of the latter as an average outcome).

Lemma 5 (Weak Law of Large Numbers)**.** `P` is a distribution (over `A`). `Xs` are `n+1` identically distributed RVs (each over `A`, they all have distribution `P`). We note μ their common expected value and σ^2 their common variance. Let `X` be the sum of the `Xs` considered independent, i.e., $X =_i \Sigma\, Xs$.
20 We have:

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

The weak law of large numbers says that the average of the outcome of `X` gets closer to μ with a large `n`.

Proof. Script excerpt:

```
have HV : 𝒱 (X  /  n+1) = σ^2 / n+1.
  ...
...
/* Pr p_ X [pred t | Rabs ((1 * / n+1  ×  X) t - μ) ≥ ε] ≤
   𝒱 (X  /  n+1) * / ε ^ 2 */
have HE : ℰ (X  /  n+1) = μ.
  ...
rewrite -{}HE.
/* Pr p_ X [pred t | Rabs ((1 * / n+1  ×  X) t - ℰ (X  /  n+1)) ≥ ε] ≤
   𝒱 (X  /  n+1) * / ε ^ 2 */
have cheby : Pr p_(X  /  n+1)
  [pred t | Rabs (X t / n+1 - ℰ (X  /  n+1)) ≥ ε] ≤ 𝒱 (X  /  n+1) / ε ^2.
  ...
```

□

2 Information Theory

We now come to the definition of *typical sequence*. Let us assume that we are given a source that emits symbols. Intuitively, a typical sequence is a `n`-tuple of symbols that is expected to be observed when `n` is "large". For example, a tuple emitted by a binary source that emits 0 with probability 2/3 is typical when it contains about two thirds of 0s.

21 The definition of typical sequences relies on the definition of *entropy*.

Definition 10 (Entropy)**.** The entropy of a distribution `P` (over `A`) is noted `𝓗P` and is defined as `- Σ_(a in A) P a * log(P a)`.

The entropy is a measure of uncertainty. For instance, a source that always emits the same symbol has entropy 0. A source that emits symbols drawn uniformly from an alphabet of cardinal n is maximal: $\log n$.

Definition 11 (Typical Sequence)**.** Let `P` be a distribution over `A`. A `n`-tuple `x` is a `P`,ε-typical

22 sequence when it satisfies the following predicate:

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

23

Definition 12 (Typical Set)**.** Given a natural `n`, a distribution `P`, and an ε, we note `𝒯𝒮 P n ε` the set of typical sequences, i.e., the set `[set x | typ_seq x]`.

(`[set x | ...]` is SSReflect notation for finite sets.)

To establish that typical sequences do have the properties we expect intuitively, we first need to prove the property of *asymptotic equipartition property* (AEP). This is a property about the outcome of several RVs that are independent and identically distributed. Informally, the AEP states that, in terms of probability, `- (1 / n+1) * log(P^n+1 x)` is "close" to the entropy `𝓗P`.

Lemma 6 (Asymptotic Equipartition Property)**.** We are given a source that emits symbols with a distribution `P` (over `A`) and some ε (`0 < ε`). We first define the following quantities that will be used as bounds hereafter:

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

24 The AEP states the following technical property about the probability of emitting a `n+1`-tuple:

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

Proof. We change the shape of the rhs of the goal to match the weak law of large numbers:

```
apply Rle_trans with (aep_σ² P / (n+1 * ε ^ 2)); last first.
  ...
/* Pr P^n+1 [pred t | 0 < P^n+1 t &
     Rabs ((− log (P^n+1) / n+1) t - ℋ P) ≥ ε] ≤
  aep_σ² P / (n+1 * ε ^ 2) */
```

25 We prepare several identically distributed RVs `Xs` with distribution `P`:

```
Definition map_mlog A n (P : dist A) : n.-tuple (rvar A) :=
  [tuple of map (fun X ⇒ − log X) (nseq n P)].
```

26 and the RV defined as follows:

```
Definition sum_map_mlog A (P : dist A) n : rvar [finType of n.-tuple A].
apply (Build_rvar (P^n)).
exact: (fun x ⇒ Σ_(i < n) - log (P x_i)).
Defined.
```

We can show that the latter is the sum of the former (considered as independent), i.e. `sum_map_mlog P n+1` $=_i \Sigma$ `map_mlog n+1 P`.

```
move: (mlog_tuple_isum_map_mlog P n) ⇒ Hisum.
/* Hisum : sum_map_mlog P n+1 =ᵢ Σ map_mlog n+1 P */
```

The conclusion comes from the application of the weak law of large numbers:

```
move: (wlln (@E_map_mlog _ P n) (@V_map_mlog _ P n) Hisum Hε) ⇒ law_large.
/* law_large : Pr p_ (sum_map_mlog P n+1)
     [pred t | Rabs ((sum_map_mlog P n+1 / n+1) t - ℋ P) ≥ ε] ≤
   aep_σ² P / (n+1 * ε ^ 2) */
eapply Rle_trans; last by apply law_large.
/* Pr P^n+1 [pred t | 0 < P^n+1 t &
     Rabs ((− log (P^n+1) / n+1) t - ℋ P) ≥ ε] ≤
   Pr p_ (sum_map_mlog P n+1)
     [pred t | Rabs ((sum_map_mlog P n+1 / n+1) t - ℋ P) ≥ ε] */
apply Pr_incl ⇒ i /=.
...
```

□

27 **Lemma 7** (Typical Set Non-empty)**.** When `n` is large enough, $\mathcal{TS}$ is not empty:

```
Definition TS_0 (H : aep_bound P ε ≤ n+1) : [finType of n+1.-tuple A]. ...
Lemma TS_0_is_typ_seq (k_k0 : aep_bound P ε ≤ n+1) : TS_0 k_k0 ∈ 𝒯𝒮 P n+1 ε.
```

Lemma 8 (Typical Set Upper-bound)**.** The cardinal of a P,n,ε-typical set is upper-bounded as follows: 28

```
Lemma TS_sup : | 𝒯𝒮 P n ε | ≤ exp (n * (ℋ P + ε)).
```

Lemma 9 (Probability of Typicality)**.** When n gets large, emitted sequences are more likely to be typical: 29

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

3 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 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 for compression with negligible error-rate.

30 **Definition 13** (Source Code)**.** A k,n-source code from alphabet A is a pair of functions:

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

The main characteristic of a source code is its rate:

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

The rate is the number of bits needed per symbol.

Definition 14 (Error Rate)**.** Let us assume a source of symbols with probability P (over A). The error rate of a k,n-source code sc is noted $\bar{e}_{src}$(P , sc) and is defined as: 31

```
Pr (P^k) [pred x | dec sc (enc sc x) ≠ x].
```

Theorem 1 (Source Coding—Direct Part)**.** Given a source of symbols from the alphabet A with distribution P, there exist source codes of rate r $\in \mathcal{Q}^+$ larger than the entropy $\mathcal{H}$P such that the error rate can be made arbitrary small: 32

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

Proof. The first step is to instantiate k and n. For that purpose, we introduce the following intermediate constants: 33

```
Let r := num / den+1.
Definition ε := Rmin (r - ℋ P) λ.
Definition δ := Rmax (aep_bound P (ε / 2)) (2 / ε).
```

34 We instantiate `k` such that $\delta \leq$`k`; `n` follows by the definition of the rate of the source code: w

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

Let k' := sval (SrcDirectBound den δ).
Definition k := k'+1 * den+1.
Definition n := k'+1 * num.
```

35 We now construct a generic `k`,`n`-source code encoder. Let `S` be a subset of `A`$^{k+1}$ such that its cardinal is strictly less than 2^n. The function that associates to the `i`th element `x` of `S` the `n`-bit encoding of `i+1` in binary and `0...0` (`n` times) otherwise is defined as follows:

```
Definition f : encT A k+1 n := fun x ⇒
  if x ∈ S then
    let i := index x (enum S) in Tuple (size_nat2bin_b i+1 n)
  else
    [tuple of nseq n false].
```

36 We now construct a corresponding `k`,`n`-source code decoder. Given a `n`-tuple `x`, if `x` is not 0 and smaller than `|S|`, it returns the "`x`-1"th element of `S`, and some default element otherwise:

```
Variable def : k+1.-tuple A.    Hypothesis Hdef : def ∈ S.

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.
```

37 By construction:

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

38 We now instantiate the generic source code above so that it encodes properly the typical sequences (proof script excerpt):

```
set S := 𝒯𝒮 P k (ε / 2).
set def := TS_0 halfepsilon0 halfepsilon1 k_k0.
set F := f n S.
set PHI := @ϕ _ n _ S def.
exists {| enc := F; dec := PHI |}.
```

The rate of the instantiated source code is the good one by construction. The conclusion of the proof follows from the properties of typical sequences:

`/* goal(1):` $\bar{e}_{src}$`(P, {| enc := F; dec := PHI |}) ≤ λ */`

The lhs of `goal(1)` can be rewritten so as to transform the goal into:

```
/* goal(2): 1 - Pr P^k [pred t in 𝒯𝒮 P k (ε / 2)] ≤ λ */
```

The latter is easily provable using the lemma `Pr_TS_1` (Lemma 9) because it can be rewritten as follows:

```
/* Pr P^k [pred t in 𝒯𝒮 P k (ε / 2)] ≥ 1 - ε / 2 */
```

We therefore only need to prove the rewriting from `goal(1)` to `goal(2)`, i.e.:

```
/* Σ_(t|PHI (F t) ≠ t) Π_(i<k) P t_i = 1 - Σ_(t|t ∈ 𝒯𝒮 P k (ε / 2)) Π_(i<k) P t_i */
```

By appealing to `Pr_cplt` (probability of the complement event, Lemma 1), the rhs can be rewritten such that:

```
/* Σ_(t|PHI (F t) ≠ t) Π_(i<k) P t_i = Σ_(t|(t ∈ k.-tuple A) ∧ (t ∉ S)) P^k t */
```

which amounts to prove:

```
/* (PHI (F t) ≠ t) = (t ∉ S) */
```

This is essentially the lemma `ϕ_f` but we need to prove two things to use it. The first one is `def ∈ S`, that comes directly from `TS_0_is_typ_seq` (see Lemma 7). The second one is `| S | <2 ^ n`. This can be reduced to `1 + |S| ≤exp(k * (ℋP + ε))` by using the definition of the source rate and the definition of ε. This can be further reduced to `1 + |S| ≤exp(1 + k * (ℋP + ε/ 2))` which can be proven using `TS_sup` (Lemma 8). □

39 **Theorem 2** (Source Coding—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 : ∀ λ, 0 < λ < 1 →
  ∀ r : 𝒬⁺, 0 < r < H P →
    ∀ n k (sc : scode A k+1 n), r = SrcRate sc →
      SrcConverseBound P (num r) (den r) n λ ≤ k+1 →
     ē_src(sc , P) ≥ λ.
```

40 where the bound `SrcConverseBound` gives a precise meaning to the claim that would otherwise be informally summarized as "for `k` big enough":

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

See [2] for the proof and also for the channel coding theorems.

References

[1] Reynald Affeldt, Manabu Hagiwara. Formalization of Shannon's Theorems in SSReflect-Coq. In 3rd Conference on Interactive Theorem Proving (ITP 2012), volume 7406 of Lecture Notes in Computer Science, pages 233-249. Springer, August 2012

[2] Reynald Affeldt, Manabu Hagiwara, Jonas Sénizergues. Formalization of Shannon's Theorems. `http://staff.aist.go.jp/reynald.affeldt/shannon`. Coq documentation. AIST, 2012

[3] Kenneth Appel, Wolfgang Haken. Every map is four colourable. Bulletin of the American Mathematical Society 82:711–712 (1976)

[4] The Coq Proof Assistant. `http://coq.inria.fr` INRIA, 1985–2012

[5] Georges Gonthier, Assia Mahboubi, Enrico Tassi. A Small Scale Reflection Extension for the Coq System. Technical Report 6455. Version 11. INRIA, 2012

[6] Georges Gonthier. RE: [Coqfinitgroup-commits] r4105 - trunk. Eletronic mail. Available at: `http://www.msr-inria.inria.fr/events-news/feit-thompson-proved-in-coq`.

[7] Thomas Hales. A proof of the Kepler conjecture. Annals of Mathematics 162(3):1065–1185 (2005)

[8] Jean van Heijenoort. From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931. Harvard University Press, 1967

[9] Manabu Hagiwara. Coding Theory: Mathematics for Digital Communication. In Japanese. `http://www.nippyo.co.jp/book/5977.html`. Nippon Hyoron Sha, 2012