One Part of Shannon’s Source Coding Theorem

Quentin Hibon

February 23, 2021

Abstract
This document contains a proof of the necessary condition on the code rate of a source code, namely that this code rate is bounded by the entropy of the source. This represents one half of Shannon’s source coding theorem, which is itself an equivalence.

This proof is taken directly from the textbook [1], and transcribed rather literally into Isabelle. It is thus easier to keep the textbook proof in mind to understand this formal proof.

Contents

1 Basic types
2 Locale for the source coding theorem
3 Source coding theorem, direct: the entropy is a lower bound of the code rate
   3.1 The letter set
   3.2 Codes and words
   3.3 Related to the Kraft theorem
   3.4 Inequality of the kraft sum (source coding theorem, direct)
      3.4.1 Sum manipulation lemmas and McMillan theorem
      3.4.2 Technical lemmas about the logarithm
      3.4.3 KL divergence and properties

theory Source-Coding-Theorem
imports HOL-Probability.Information
begin

1 Basic types

type-synonym bit = bool
type-synonym bword = bit list
type-synonym letter = nat
type-synonym 'b word = 'b list

type-synonym 'b encoder = 'b word ⇒ bword
type-synonym 'b decoder = bword ⇒ 'b word option

2 Locale for the source coding theorem

locale source-code = information-space +
fixes fi :: 'b ⇒ real
fixes X :: 'a ⇒ 'b

assumes distr-i: simple-distributed M X fi
assumes b-val: b = 2

fixes enc::'b encoder
fixes dec::'b decoder
assumes real-code:
dec (enc x) = Some x
enc w = [] ⟷ w = []
x ≠ [] ⟷ enc x = enc [hd x] @ enc (tl x)

3 Source coding theorem, direct: the entropy is a lower bound of the code rate

context source-code
begin

3.1 The letter set

definition L :: 'b set where
L ≡ X · space M

lemma fin-L: finite L
using L-def distr-i
by auto

lemma emp-L: L ≠ {}
using L-def subprob-not-empty
by auto

3.2 Codes and words

abbreviation real-word :: 'b word ⇒ bool where
real-word w ≡ (set w ⊆ L)

abbreviation k-words :: nat ⇒ ('b word) set where
k-words k ≡ {w. length w = k ∧ real-word w}
lemma \textit{rv-tail}:
assumes real-word \textit{w}
shows \textit{w} = [] \lor real-word (tl w)
by (meson assms list.set-sel(2) subset-code(1))
definition code-word-length :: 'e encoder \Rightarrow 'e \Rightarrow nat where
code-word-length \textit{e} \textit{l} = length (\textit{e} \textit{l})
abbreviation \textit{cw-len} :: 'b \Rightarrow nat where
cw-len \textit{l} \equiv code-word-length enc \textit{l}
definition code-rate :: 'e encoder \Rightarrow ('a \Rightarrow 'e) \Rightarrow real where
code-rate \textit{e} \textit{Xo} = \textit{expectation} (\lambda \textit{a}. (code-word-length \textit{e} ((\textit{Xo} \textit{a}))))
lemma \textit{fi-pos}:
\textit{i} \in \textit{L} \Rightarrow 0 \leq \textit{fi} \textit{i}
using simple-distributed-nonneg[of distr-i] L-def by auto
lemma (in prob-space) \textit{simp-exp-composed}:
assumes \textit{X}: simple-distributed \textit{M} \textit{X} \textit{Px}
show \textit{expectation} (\lambda \textit{a}. \textit{f} \textit{(X} \textit{a})) = \sum_{\textit{x} \in \textit{X’space} \textit{M}} \textit{f} \textit{x} \ast \textit{Px} \textit{x}
using distributed-integral[of \textit{X}, of \textit{f}]
simple-distributed-nonneg[of \textit{X}]
lebesgue-integral-count-space-finite[of \textit{X}, of \lambda \textit{x}. \textit{f} \textit{x} \ast \textit{Px} \textit{x}]
by (simp add: ac-simps)
lemma \textit{cr-rw}:
code-rate \textit{enc} \textit{X} = \sum_{\textit{i} \in \textit{X space} \textit{M}} \textit{fi} \textit{i} \ast \textit{cw-len} \textit{i}
using simp-exp-composed[of distr-i, of \textit{cw-len}]
by (simp add: mult.commute code-rate-def)
abbreviation \textit{cw-len-concat} :: 'b word \Rightarrow nat where
cw-len-concat \textit{w} \equiv \textit{foldr} (\lambda \textit{x} \textit{s}. (\textit{cw-len} \textit{x}) + \textit{s}) \textit{w} 0
lemma \textit{cw-len-length}:
cw-len-concat \textit{w} \equiv \textit{length} (\textit{enc} \textit{w})
proof (induction \textit{w})
case Nil
show \textit{?case} using real-code by simp
case (Cons \textit{a} \textit{w})
have \textit{cw-len-concat} (\textit{a} \# \textit{w}) = \textit{cw-len} \textit{a} + \textit{cw-len-concat} \textit{w} by simp
thus \textit{?case} using code-word-length-def real-code Cons
by (metis length-append list.distinct(1) list.sel(1) list.sel(3))
qed
lemma \textit{maj-fold}:
assumes \A \textit{l}. \textit{l} \in \textit{L} \Rightarrow \textit{f} \textit{l} \leq \textit{bound}
assumes real-word \textit{w}
show \textit{foldr} (\lambda \textit{x} \textit{s}. \textit{f} \textit{x} + \textit{s}) \textit{w} 0 \leq \textit{length} \textit{w} \ast \textit{bound}
using \textit{assms}
by (induction w) (simp, fastforce)

**definition** `max-len :: nat where`

`max-len = Max ((λx. cw-len x) ∘ L)`

**lemma** `max-cw`:

`l ∈ L ⇒ cw-len l ≤ max-len`

by (simp add: `max-len-def fin-L`)

### 3.3 Related to the Kraft theorem

**definition** `K :: real where`

`K = (∑ i ∈ L. 1 / b ^ (cw-len i))`

**lemma** `pos-cw-len`:

`0 < 1 / b ^ cw-len i` using `b-gt-1` by simp

**lemma** `K-pos`:

`0 < K` using `emp-L fin-L pos-cw-len sum-pos K-def` by metis

**lemma** `K-pow`:

`K = (∑ i ∈ L. 1 / b ^ (cw-len i))` using `powr-realpow b-gt-1` by (simp add: `K-def`)

**lemma** `k-words-rel`:

`k-words (Suc k) = {w. (hd w ∈ L ∧ tl w ∈ k-words k ∧ w ≠ [])}`

**proof**

fix `k`

show `k-words (Suc k) ⊆ {w. (hd w ∈ L ∧ tl w ∈ k-words k ∧ w ≠ []) }` (is `?l ⊆ ?r`)

proof

fix `w`

assume `w-kw: w ∈ k-words (Suc k)`

hence `real-word w` by simp

hence `hd w ∈ L`

by (metis (mono-tags) `w-kw hd-in-set list.size(3)` mem-Collect-eq nat.distinct(1) subset-code(1))

moreover have `length w = Suc k` using `w-kw` by simp

moreover hence `w ≠ []` by auto

moreover have `real-word (tl w)` using `(real-word w)` by auto

by auto

ultimately show `w ∈ ?r` using `w-kw` by simp

qed

next

fix `k`

show `k-words (Suc k) ⊇ {w. (hd w ∈ L ∧ tl w ∈ k-words k ∧ w ≠ [])}`

proof

fix `w`

assume `asm: w ∈ {w. hd w ∈ L ∧ tl w ∈ {w. length w = k ∧ real-word w} ∧`
\[ w \neq [] \]

hence \( hd w \in L \land \text{length}(tl w) = k \land \text{real-word}(tl w) \) by simp

hence \( \text{real-word} w \)

by (metis empty-iff insert-subset list.collapse list.set(1) set-simps(2) subsetI)

moreover hence \( \text{length} w = \text{Suc} k \) using asm by auto

ultimately show \( w \in k\text{-words}(\text{Suc} k) \) by simp

qed

qed

lemma bij-k-words:
shows \( \text{bij-betw}(\lambda wi. \text{Cons}(\text{fst} wi) (\text{snd} wi)) (L \times k\text{-words} k) (k\text{-words}(\text{Suc} k)) \)

unfolding bij-betw-def

proof

fix \( k \)

let \( ?f = (\lambda wi. \text{Cons}(\text{fst} wi)(\text{snd} wi)) \)

let \( ?S = L \times (k\text{-words} k) \)

let \( ?T = k\text{-words}(\text{Suc} k) \)

show \( \text{inj-on} ?f ?S \) by simp add: inj-on-def

show \( \text{inj-on} ?f ?T \) by (simp add: inj-on-def)

proof (rule ccontr)

assume \( ?f \cdot ?S \neq ?T \)

hence \( \exists w. w \in ?T \land w \notin ?f \cdot ?S \) by auto

then obtain \( w \) where asm: \( w \in ?T \land w \notin ?f \cdot ?S \) by blast

hence \( w = ?f(hd w, tl w) \) using k-words-rel by simp

moreover have \( (hd w, tl w) \in ?T \) using k-words-rel asm by simp

ultimately have \( w \in ?T \) using k-words-rel asm by simp

thus \( False \) using asm by simp

qed

qed

lemma finite-k-words: finite \( (k\text{-words} k) \)

proof (induct \( k \))

case \( 0 \)

show \( \text{?case} \) by simp

case (Suc \( n \))

thus \( \text{?case} \) using bij-k-words bij-betw-finite fin-L by blast

qed

lemma cartesian-product:

fixes \( f :: (c \Rightarrow \text{real}) \)

fixes \( g :: (d \Rightarrow \text{real}) \)

assumes \( \text{finite} A \)

assumes \( \text{finite} B \)

shows \( \sum_{b \in B} g \cdot b = (\sum_{a \in A} f \cdot a) = (\sum_{ab \in A \times B} (\text{fst} ab) \cdot f \cdot (\text{snd} ab)) \)

using bilinear-times bilinear-sum[where \( h=(\lambda x. x \cdot y) \) and \( f=f \) and \( g=g \)]

assms

by (metis (erased, lifting) sum.cong split-beta Groups.absemigroup-mult-class.mult.commute)

lemma \( K\text{-power} \):
shows $K^k = (\sum w \in (k\text{-words } k). \frac{1}{b^{cw\text{-len-concat } w}})$

proof (induct $k$)

  case $0$
  have $k\text{-words } 0 = \{[]\}$ by auto
  thus $?case$ by simp

next

  case (Suc $n$)
  have $K^\text{Suc } n = K^\text{Suc } n * K$ by simp
  also have $\ldots = (\sum w \in k\text{-words } n. \frac{1}{b^{cw\text{-len-concat } w}}) * (\sum i \in L. \frac{1}{b^{cw\text{-len } i}})$
    using Suc.hyps $K$-def by auto
  also have $\ldots = (\sum wi \in L \times k\text{-words } n. \frac{1}{b^{cw\text{-len-concat } (\text{snd } wi)} + cw\text{-len } (\text{fst } wi)})$
    by (metis (no-types, lifting) power-add add.commute power-one-over)
  also have $\ldots = (\sum wi \in L \times k\text{-words } n. \frac{1}{b^{cw\text{-len-concat } (\text{fst } wi) * \text{snd } wi}})$
    using fin-L finite-k-words cartesian-product by blast
  also have $\ldots = (\sum wi \in L \times k\text{-words } n. \frac{1}{b^{cw\text{-len-concat } (\text{fst } wi * \text{snd } wi})})$
    by (metis (erased, lifting) add.commute comp-apply foldr.simps(2))
  also have $\ldots = (\sum w \in (k\text{-words } (\text{Suc } n)). \frac{1}{b^{cw\text{-len-concat } w}})$
    using bij-k-words sum.reindex-bij-betw by fastforce
  finally show $?case$ by simp

qed

lemma bound-len-concat:
shows $w \in k\text{-words } k \Rightarrow cw\text{-len-concat } w \leq k * \text{max-len}$
  using max-cw maj-fold by blast

3.4 Inequality of the Kraft sum (source coding theorem, direct)

3.4.1 Sum manipulation lemmas and McMillan theorem

lemma sum-vimage-proof:
fixes $g\::\::\text{nat} \Rightarrow \text{real}$
assumes $\forall w. f w < bd$
shows finite $S \Rightarrow (\sum w \in S. g (f w)) = (\sum m=0..<bd. (\text{card } ((f-\{m\}) \cap S) ) * g m )$
(is $- \Rightarrow - = (\sum m=0..<bd. ??? m S)$)

proof (induct $S$ rule: finite-induct)

  case empty
  show $?case$ by simp

next

  case (insert $x F$)
  let $??r = (\sum m = 0..<bd. ??? m (\text{insert } x F))$
  have $(f x) \in \{0..<bd\}$ using assms by simp
  hence $\forall h::(\text{nat} \Rightarrow \text{real}). (\sum m=0..<bd. h m) = (\sum y \in (\{0..<bd\} - \{f x\}). h y) + h (f x)$
by (metis diff-add-cancel finite-atLeastLessThan sum-diff1)
moreover hence
\( \sum m = 0..<bd. \ ?ff (insert x F) \)
\( = (\sum m\in\{0..<bd\} - \{f x\}. \ ?ff (insert x F)) + \text{card} (f -\{f x\} \cap F) \ast g (f x) \)
by (simp add: semiring-normalization-rules(2), simp add: insert)
ultimately have \( \sum m = 0..<bd. \ ?ff (insert x F) \)
\( = (\sum m\in\{0..<bd\}. \ ?ff m \ ) + g (f x) \)
by fastforce
thus ?case using insert by simp
qed

lemma sum-vimage:
fixes g::nat \Rightarrow real
assumes bounded:\( \forall w. \ w \in S \rightarrow f w < bd \) and \( 0 < bd \)
assumes finite: finite S
shows \( (\sum w\in S. \ g (f w)) = (\sum m=0..<bd. \ (\text{card} ((f -\{m\}) \cap S) ) \ast g m) \)
(is ?s1 = ?s2)
proof –
let ?ff = (\lambda x. if x\in S then f x else 0)
let ?ss1 = (\sum w\in S. \ g (\ ?ff w))
let ?ss2 = (\sum m=0..<bd. \ (\text{card} ((\ ?ff -\{m\}) \cap S) ) \ast g m)
have ?s1 = ?ss1 by simp
moreover have \( \forall m. \ ?ff -\{m\} \cap S = f -\{m\} \cap S \) by auto
moreover hence ?s2 = ?ss2 by simp
moreover have \( \forall w. \ ?ff w < bd \) using assms by simp
moreover hence ?ss1 = ?s2 using sum-vimage-proof[\text{of} \ ?ff \] finite by blast
ultimately show ?s1 = ?s2 by metis
qed

lemma K-rw:
\( \sum w\in (k\text{-words } k). \ 1 / b^ (cw\text{-len-concat } w)) = (\sum m=0..< Suc (k\ast \text{max-len}). \ \text{card} (k\text{-words } k \cap ((cw\text{-len-concat }) -\{m\}) \ast (1 / b ^\{m\})) \) (is ?L = ?R)
proof –
have \( \forall w. \ w \in k\text{-words } k \rightarrow cw\text{-len-concat } w < Suc (k \ast \text{max-len}) \)
by (simp add: bound-len-concat le-imp-less-Suc)
moreover have \( ?R = (\sum m = 0..< Suc (k \ast \text{max-len}). \ (\text{card} (cw\text{-len-concat }) -\{m\} \cap k\text{-words } k) \ast (1 / b ^\{m\})) \) (is \ ?L = \ ?R)
by (metis Int-commute)
moreover have \( 0 < Suc (k\ast \text{max-len}) \) by simp
ultimately show \( ?R \) using finite-k-words
sum-vimage[\text{where} f=cw\text{-len-concat} \text{ and } g = \lambda i. \ 1/ (b ^\{i\})]
by fastforce
qed

definition set-of-k-words-length-m :: nat \Rightarrow nat \Rightarrow \text{"b word set where}
set-of-k-words-length-m \( k m = \{ xk. xk \in k\text{-words} k \} \cap (cw\text{-}concat) - \{m\} \)

lemma am-inj-code: inj-on enc \(((cw\text{-}concat) - \{m\}\)) (is inj-on - ?s) using inj-on-def[of enc ?s] real-code
by (metis option.inject)

lemma img-inc: enc\('cw\text{-}len\text{-}concat' - \{m\}\) \( \subseteq \{ bl. length bl = m \} \) using cw-len-length by auto

lemma bool-lists-card: \( \text{card} \{ bl::bool list. length bl = m \} = b^m \)
using card-lists-length-eq[of UNIV::bool set] by (simp add: b-val)

lemma bool-list-fin: finite \( \{ bl::bool list. length bl = m \} \)
using finite-lists-length-eq[of UNIV::bool set] by (simp add: b-val)

lemma set-of-k-words-bound:
shows \( \text{card} (\text{set-of-k-words-length-m} k m) \leq b^m \) (is \(?c \leq \?b\))
proof –
have \( \text{card-w-len-m-bound: card} \ (cw\text{-}len\text{-}concat - \{m\}) \leq b^m \)
by (metis (no-types, lifting) am-inj-code bool-list-fin bool-lists-card card-image card-mono
img-inc of-nat-le-iff)

have \( \text{set-of-k-words-length-m} k m \leq (cw\text{-}len\text{-}concat) - \{m\} \)
by (simp add: set-of-k-words-length-m-def)

hence \( \text{card} \ (\text{set-of-k-words-length-m} k m) \leq \text{card} \ ((cw\text{-}len\text{-}concat) - \{m\}) \)
by (metis (no-types, lifting) am-inj-code bool-list-fin card.infinite card-0-eq card-image card-mono empty-iff finite-subset img-inc inf-img-fin-dom)

thus \(?\text{thesis}\) using card-w-len-m-bound by simp
qed

lemma empty-set-k-words:
assumes \( 0 < k \)
shows \( \text{set-of-k-words-length-m} k 0 = \{\} \)
proof(rule ccontr)
assume \( \neg \text{set-of-k-words-length-m} k 0 = \{\} \)
hence \( \exists x. x \in \text{set-of-k-words-length-m} k 0 \) by auto
then obtain \( x \) where \( x\text{-def: } x \in \text{set-of-k-words-length-m} k 0 \) by auto
hence \( x \neq [] \) unfolding set-of-k-words-length-m-def using assms by auto
moreover have \( cw\text{-}len\text{-}concat \ (hd x\#tl x) = cw\text{-}len\text{-}concat \ (tl x) + cw\text{-}len \ (hd x) \)
by (metis add.commute comp-apply foldr.simps(2))
moreover have \( \text{enc} \ [(hd x)] \neq [] \) using assms real-code by blast
moreover hence \( 0 < cw\text{-}len \ (hd x) \) unfolding code-word-length-def by simp
ultimately have \( x \notin \text{set-of-k-words-length-m} k 0 \) by (simp add: set-of-k-words-length-m-def)
thus \( False \) using x-def by simp
qed
lemma $K$-rw2:
assumes $0 < k$
shows $(\sum_{m=0..<\text{Suc}(k * \text{max-len})} \text{card (set-of-k-words-length-m k m)} / b^m) \leq (k * \text{max-len})$
proof
  have $(\sum_{m=1..<\text{Suc}(k * \text{max-len})} \text{card (set-of-k-words-length-m k m)} / b^m) \leq (\sum_{m=1..<\text{Suc}(k * \text{max-len})} b^m / b^m)$
    using set-of-k-words-bound b-val Groups-Big.sum-mono[of \{1..<\text{Suc}(k * \text{max-len})\}]
    (\lambda m. (\text{card (set-of-k-words-length-m k m)}) / b^m) \lambda m. b^m / b^m]
    by simp
  moreover have $(\sum_{m=1..<\text{Suc}(k * \text{max-len})} b^m / b^m) = (\sum_{m=1..<\text{Suc}(k * \text{max-len})} 1)$
    using b-gt-1 by simp
  moreover have $(\sum_{m=1..<\text{Suc}(k * \text{max-len})} 1) = (k * \text{max-len})$
    by simp
  ultimately have $(\sum_{m=1..<\text{Suc}(k * \text{max-len})} \text{card (set-of-k-words-length-m k m)} / b^m) \leq k * \text{max-len}$
    by (metis One-nat-def card-atLeastLessThan card-eq-sum diff-Suc-Suc real-of-card)
  thus \thesis using empty-set-k-words assms by (simp add: sum-shift-lb-Suc0-0-upt split: if-split-asm)
qed

lemma $K$-power-bound:
assumes $0 < k$
shows $K \leq k * \text{max-len}$
using assms $K$-power $K$-rw $K$-rw2
by (simp add: set-of-k-words-length-m-def)

theorem McMillan:
shows $K \leq 1$
proof
  have ineq: $\forall k. 0 < k \implies K \leq \text{root k k * root k max-len}$
    using $K$-pos $K$-power-bound
    by (metis (no-types, hide-lams) not-less of-nat-0-le-iff of-nat-mult power-strict-mono
      real-root-mul real-root-pos-pos-le real-root-pos-unique real-root-power)
  hence $0 < \text{max-len} \implies (\lambda k. \text{root k k * root k max-len}) \longrightarrow 1$
    by (auto intro!: tendsto-eq-intros LIMSEQ-root LIMSEQ-root-const)
  moreover have $\forall n \geq 1. K \leq \text{root n n * root n max-len}$
    using ineq by simp
  moreover have $\text{max-len} = 0 \implies K \leq 1$ using ineq by fastforce
  ultimately show $K \leq 1$ using LIMSEQ-le-const by blast
qed

lemma entropy-rw: $H(X) = -(\sum_{i \in L} \text{fi i} * \log b \text{(fi i)})$
using entropy-simple-distributed[of distr-i]
by (simp add: L-def)
3.4.2 Technical lemmas about the logarithm

lemma log-mult-ext3:
\[0 \leq x \implies 0 < y \implies 0 < z \implies x \cdot \log_b (x \cdot y \cdot z) = x \cdot \log_b (x \cdot y) + x \cdot \log_b z\]
by (cases x=0) (simp add: log-mult-eq abs-of-pos distrib-left less-eq-real-def)

lemma log-mult-ext2:
\[0 \leq x \implies 0 < y \implies x \cdot \log_b (x \cdot y) = x \cdot \log_b x + x \cdot \log_b y\]
using log-mult-ext3[where \(y=1\)] by simp

3.4.3 KL divergence and properties

definition KL-div :: \(\cdot \) set \(\mapsto \) (\(\cdot \) \(\mapsto \) real) \(\mapsto \) (\(\cdot \) \(\mapsto \) real) \mapsto real where
KL-div \(S \cdot a \cdot d\) = (\(\sum_{i \in S}. a i \cdot \log_b (a i / d i)\))

lemma KL-div-mul:
assumes \(0 < d\ d \leq 1\)
assumes \(\land i. i \in S \implies 0 \leq a i\)
assumes \(\land i. i \in S \implies 0 < e i\)
shows KL-div \(S \cdot a \cdot e\) \(\geq\) KL-div \(S \cdot a (\lambda i. e i / d)\)

unfolding KL-div-def

proof
−
\{
\{ 
  fix \(i\)
  assume \(i \in S\)
  hence \(a i / (e i / d) \leq a i / e i\) using assms
  by (metis (no-types) div-by-1 frac-le less-imp-triv not-less)
  hence \(\log_b (a i / (e i / d)) \leq \log_b (a i / e i)\) using assms(1)
  by (metis (full-types) b-gt-1 divide-divide-eq-left inverse-divide le-less-linear
log-le
  log-neg-const order-refl times-divide-eq-right zero-less-mult-iff)
\}
thus (\(\sum_{i \in S}. a i \cdot \log_b (a i / (e i / d))\)) \(\leq\) (\(\sum_{i \in S}. a i \cdot \log_b (a i / e i)\))
  by (meson mult-left-mono assms sum-mono)
\}
qed

lemma KL-div-pos:
fixes \(a\ e::\cdot \) \(\mapsto \) real
assumes fin: finite \(S\)
assumes nemp: \(S \neq \{\}\)
assumes non-null: \(\land i. i \in S \implies 0 < a i\ \land i. i \in S \implies 0 < e i\)
assumes sum-a-one: \((\sum_{i \in S}. a i) = 1\)
assumes sum-c-one: \((\sum_{i \in S}. e i) = 1\)
shows \(0 \leq KL-div \(S \cdot a \cdot e\)\)

unfolding KL-div-def

proof
−
let \(?f = \lambda i. e i / a i\)
have \(f\)-pos: \(\land i. i \in S \implies 0 < ?f i\)
  using non-null
  by simp
have \( a \)-pos: \( \forall i. i \in S \implies 0 \leq a \cdot i \)
using non-null
by (simp add: order.strict-implies-order)

have \( -\log b \left( \sum i \in S. a \cdot i \cdot e \cdot i / a \cdot i \right) \leq \left( \sum i \in S. a \cdot i \cdot \log b \ (e \cdot i / a \cdot i) \right) \)
using convex-on-sum\{OF fin nemp minus-log-convex[OF b-gt-1] convex-real-interval(3)
sum-a-one a-pos, of \( \lambda i. e \cdot i / a \cdot i \) \} f-pos by simp
also have \( -\log b \left( \sum i \in S. a \cdot i \cdot e \cdot i / a \cdot i \right) = -\log b \left( \sum i \in S. e \cdot i \right) \)
proof –
from non-null(\( f \)) have \( \forall i. i \in S \implies a \cdot i \cdot e \cdot i / a \cdot i = e \cdot i \) by force
thus \?thesis by simp

qed

lemma \( KL\text{-}div\text{-}pos\text{-}emp \):
\( 0 \leq KL\text{-}div \ \{ a \} \ e \) by (simp add: \( KL\text{-}div\text{-}def \))

lemma \( KL\text{-}div\text{-}pos\text{-}gen \):
fixes \( a \ d:\cdot \ b \Rightarrow \text{real} \)
assumes fin: \( \text{finite} \ S \)
assumes non-null: \( \forall i. i \in S \implies 0 < a \cdot i \) \( \forall i. i \in S \implies 0 < d \cdot i \)
assumes sum-a-one: \( \sum i \in S. a \cdot i \) = \( 1 \)
assumes sum-d-one: \( \sum i \in S. d \cdot i \) = \( 1 \)
shows \( 0 \leq KL\text{-}div \ S \ a \ d \)
using \( KL\text{-}div\text{-}pos \ KL\text{-}div\text{-}pos\text{-}emp \) assms by metis

theorem \( KL\text{-}div\text{-}pos2 \):
fixes \( a \ d:\cdot \ b \Rightarrow \text{real} \)
assumes fin: \( \text{finite} \ S \)
assumes non-null: \( \forall i. i \in S \implies 0 \leq a \cdot i \) \( \forall i. i \in S \implies 0 < d \cdot i \)
assumes sum-a-one: \( \sum i \in S. a \cdot i \) = \( 1 \)
assumes sum-c-one: \( \sum i \in S. c \cdot \cdot i \) = \( 1 \)
shows \( 0 \leq KL\text{-}div \ S \ a \ d \)
proof –
have \( S = (S \cap \{ i. 0 < a \cdot i \}) \cup (S \cap \{ i. 0 = a \cdot i \}) \) using non-null(\( f \)) by fastforce
moreover have \( (S \cap \{ i. 0 < a \cdot i \}) \cap (S \cap \{ i. 0 = a \cdot i \}) = \{ \} \) by auto
ultimately have eq: \( KL\text{-}div \ S \ a \ d = KL\text{-}div \ (S \cap \{ i. 0 < a \cdot i \}) \ a \ d + KL\text{-}div \ (S \cap \{ i. 0 = a \cdot i \}) \ a \ d \)
da d
unfolding \( KL\text{-}div\text{-}def \)
by (metis (mono-tags, lifting) fin finite-Un sum.union-disjoint)
have \( KL\text{-}div \ (S \cap \{ i. 0 = a \cdot i \}) \ a \ d = 0 \) unfolding \( KL\text{-}div\text{-}def \) by simp
hence \( KL\text{-}div \ S \ a \ d = KL\text{-}div \ (S \cap \{ i. 0 < a \cdot i \}) \ a \ d \) using eq by simp
moreover have \( 0 \leq KL\text{-}div \ (S \cap \{ i. 0 < a \cdot i \}) \ a \ d \)
proof (cases (S ∩ {i. 0 < a i}) = {})  
  case True  
    thus ?thesis unfolding KL-div-def by simp  
next  
  case False  
  let ?c = λi. d i / (∑ j∈(S ∩ {i. 0 < a i}). d j)  
  have 1: (∀i. i ∈ S ∩ {i. 0 < a i} →→ 0 < a i) by simp  
  have 2: (∀i. i ∈ S ∩ {i. 0 < a i} →→ 0 < ?c i)  
    by (metis False IntD1 divide-pos-pos fin finite-Int non-null(2) sum-pos)  
  have 3: (∑ i∈ (S ∩ {i. 0 < a i}). a i) = 1  
    using sum.cong[of S, of S, of (λx. if x ∈ {i. 0 < a i} then a x else 0), of a]  
    sum.inter-restrict[OF fin, of a] non-null(1) sum-a-one  
    by fastforce  
  have (∑ i∈S ∩ {j. 0 < a j}. ?c i) = (∑ i∈S ∩ {j. 0 < a j}. d i) / (∑ i∈S ∩ {j. 0 < a j}. d i)  
    by (metis sum-divide-distrib)  
  hence 5: (∑ i∈S ∩ {j. 0 < a j}. ?c i) = 1 using 2 False by force  
  hence 0 ≤ KL-div (S ∩ {j. 0 < a j}) a ?c  
    using KL-div-pos-gen[  
      OF finite-Int[OF disjI1, of S, of {j. 0 < a j}], of a, of ?c  
    ] 1 2 3  
    by (metis fin)  
  have fstdb: 0 < (∑ i∈S ∩ {i. 0 < a i}. d i) using non-null(2) False  
    by (metis Int-Collect fin finite-Int sum-pos)  
  have 6: 0 ≤ KL-div (S ∩ {i. 0 < a i}) a (λi. d i / (∑ i∈(S ∩ {i. 0 < a i}). d i)).  
    using 2 3 5  
      KL-div-pos-gen[  
        OF finite-Int[OF disjI1, OF fin], of {i. 0 < a i}, of a, of ?c  
      ]  
      by simp  
  hence  
    KL-div (S ∩ {j. 0 < a j}) a (λi. d i / (∑ i∈(S ∩ {i. 0 < a i}). d i)). ≤  
    KL-div (S ∩ {i. 0 < a i}) a d  
    using non-null sum.inter-restrict[OF fin, of d, of {i. 0 < a i}]  
    sum-mono[of S, of (λx. if x ∈ {i. 0 < a i} then d x else 0), of d] non-null(2)  
    sum-c-one  
    non-null(2) fstdb KL-div-mul  
    by force  
  moreover have 0 ≤ KL-div (S ∩ {j. 0 < a j}) a (λi. d i / (∑ i∈(S ∩ {i. 0 < a i}). d i)).  
    using KL-div-pos-gen[ OF finite-Int[OF disjI1, OF fin]] using 2 3 5 by  
    fastforce  
    ultimately show 0 ≤ KL-div (S ∩ {j. 0 < a j}) a d by simp  
  qed  
  ultimately show ?thesis by simp  
  qed  

lemma sum-div-1:
\[ \text{proof} \]
\[ \text{shows } (\sum_{i \in A} f_1) + (\sum_{j \in A} f_2) = 1 \]
\[ \text{by (metis (no-types) assms right-inverse-eq sum-divide-distrib)} \]

\text{theorem} \ rate-lower-bound:
\text{shows } H(X) \leq \text{code-rate enc} X

\text{proof} –
\[ \text{let } \lambda r = \text{code-rate enc} X \]
\[ \text{let } \lambda r = (\lambda i. 1 / ((b \text{ powr} \cw-len i) \ast K)) \]
\[ \text{have pos-pi: } \forall i. i \in L \implies 0 \leq f_i \text{ using } f_i \text{-pos by simp} \]
\[ \{ \]
\[ \text{fix } i \]
\[ \text{assume } i \in L \]
\[ \text{hence } f_i \ast (\log b (1 / (1 / b \text{ powr} \cw-len i))) + \log b (f_i) \]
\[ = f_i \ast \log b (f_i / (1 / b \text{ powr} \cw-len i)) \]
\[ \text{using log-mult-ext2 [OF pos-pi, of } i \text{]} \text{ b-gt-1} \]
\[ \text{by simp (simp add: algebra-simps)} \]
\[ \} \]
\[ \text{hence eqi:} \]
\[ \forall i. i \in L \implies f_i \ast (\log b (1 / (1 / b \text{ powr} \cw-len i))) + \log b (f_i) \]
\[ = f_i \ast \log b (f_i / (1 / b \text{ powr} \cw-len i)) \]
\[ \text{by simp} \]
\[ \text{have sum-one-L: } (\sum_i \in L. f_i) = 1 \]
\[ \text{using simple-distributed-sum-space[OF distr-i] by (simp add: L-def)} \]
\[ \{ \]
\[ \text{fix } i \]
\[ \text{assume } i \in L \]
\[ \text{hence h1: } 0 \leq f_i \text{ using pos-pi by blast} \]
\[ \text{have h2: } 0 < K \cdot (1/b \text{ powr} \cw-len i) \text{ using } b-gt-1 \text{ K-pos by auto} \]
\[ \text{have h3: } 0 < 1 / K \text{ using K-pos by simp} \]
\[ \text{have} \]
\[ f_i \ast \log b (f_i \ast K / (1/b \text{ powr} \cw-len i) \ast (1 / K)) = \]
\[ f_i \ast \log b (f_i \ast K / (1/b \text{ powr} \cw-len i)) + f_i \ast \log b (1/K) \]
\[ \text{using log-mult-ext3[OF h1 h2 h3]} \]
\[ \text{by (metis times-divide-eq-right)} \]
\[ \} \]
\[ \text{hence big-eq:} \]
\[ \forall i. i \in L \implies f_i \ast \log b (f_i \ast K / (1/b \text{ powr} \cw-len i) \ast (1 / K)) = \]
\[ f_i \ast \log b (f_i \ast K / (1/b \text{ powr} \cw-len i)) + f_i \ast \log b (1/K) \]
\[ \text{by (simp add: inverse-eq-divide)} \]
\[ \text{have 1: } \forall r = \mathcal{H}(X) = (\sum_i \in L. f_i \ast \cw-len i) + (\sum_i \in L. \neg log b (f_i)) \]
\[ \text{using K-def entropy-rw cr-rw L-def by simp} \]
\[ \text{also have 2: } (\sum_i \in L. f_i \ast \cw-len i) = (\sum_i \in L. f_i \ast (\neg log b (1/(b \text{ powr} (\cw-len i)))) \]
\[ \text{using b-gt-1 log-divide by simp} \]
\[ \text{also have } \]
by simp
finally have
\[ \log b (\text{fi}\ i) \]
by simp
have \( \log b (\text{fi}\ i) \)
using \( b\text{-gt-1} \)
by (simp add: \( \text{distrib-left sum.distrib} \))
also have \( \log b (\text{fi}\ i / (1 / b\ \text{powr}\ \text{cw-len}\ i)) + \log b (\text{fi}\ i) \)
using \( K\text{-pos} \)
by (simp add: \( \text{distrib-right sum.distrib} \))
also have \( 1 \)
by simp
moreover have \( \wedge i. 0 < \text{fi}\ i \)
using \( b\text{-gt-1} \)
by simp
moreover have \( \sum i\in L. \text{fi}\ i = 1 \)
by simp
ultimately have \( 0 \leq KL\text{-div}\ K\text{-pos} \)
by simp
hence \( \log b (1 / K) \leq \log b (1 / K) \)
by simp
moreover from McMillan have \( 0 \leq \log b (1 / K) \)
using \( K\text{-pos} \)
by (simp add: \( b\text{-gt-1} \))
ultimately show \( \text{thesis} \)
by simp
qed
References