Class Road Map - IT-I

- L19 (1/6): Overview, Communications, Gaussian Channel
- L20 (1/8): Gaussian Channel, band limitation, parallel channels, optimization and duality
- L21 (1/13): parallel channels, colored noise, feedback, matrix inequalities
- L22 (1/15): matrix inequalities, rate distortion.
  - (1/20): Monday holiday
- L23 (1/22): rate distortion for Bernoulli, Gaussian, and Multiple Gaussians with unequal noise
- L24 (1/27): main rate distortion theorem, geometry
- L25 (1/29): computing $R(D)$
- L26 (2/3): computing $R(D)$, alternating minimization
- L27 (2/5): Kolmogorov complexity
- L28 (2/10): algorithmic randomness, universal prob.,
- L29 (2/12): universal compression, LZ compression
  - (2/17): Monday, Holiday
- L30 (2/19):
- L31 (2/24):
- L32 (2/26):
- L33 (3/3):
- L34 (3/5):
- L35 (3/10):
- L36 (3/12):


Prof. Jeff Bilmes
Read Ch. 14 in our book (Cover & Thomas, “Information Theory”).
Read Ch. 10 in our book (Cover & Thomas, “Information Theory”).
Read Ch. 17 in our book (Cover & Thomas, “Information Theory”) on matrix inequalities.
Read Ch. 9 in our book (Cover & Thomas, “Information Theory”)
Read Ch. 5 in Boyd and Vandenberghe’s Convex Optimization book
Read all readings assigned in EE514a, Fall 2013. (see later lectures on our previous web page (http://j.ee.washington.edu/~bilmes/classes/ee514a_fall_2013/)).
No current outstanding HW.
No class Monday, Feb 17th

Office hours on Mondays, 3:30-4:30.

As always, email me if you want to skype/google hangout rather than come to office hours, also at different times.
On Final Presentations

- Your task is to give a 10-15 minute presentation that summarizes 2-3 related and significant papers that come from IEEE Transactions on Information Theory (or a very related area).
- The papers must not be ones that we covered in class, although they can be related.
- You need to do the research to find the papers yourself (i.e., that is part of the assignment).
- The majority of the papers must have been published in the last 10 years (so no old or classic papers).
- Your grade will be based on how clear, understandable, and accurate your presentation is (and also milestones).
- This is a real challenge and will require significant work! Many of the papers are complex. To get a good grade, you will need to work very hard to present very complex ideas in an extremely simple yet still precise way.
- Again, don’t expect this to be easy, you might need to try a few topics until you find one that is suitable.
Final Presentation Milestones

All submissions done in PDF file format via our assignment dropbox (https://canvas.uw.edu/courses/880971/assignments)

- **Monday, Feb 17th, 11:45pm**: Candidate proposed papers submitted. Include short at most 1-page writeup: 1) why you chose these papers; 2) how they are related to each other; 3) why they are important to pure IT; and 4) how they are fundamental and/or deep, and 5) how will you summarize them in a simple and precise way.

- **Monday, Feb 24th 11:45pm**: Updated list of proposed papers decided, based on feedback. Updated writeup with more description.

- **Monday, March 3rd 11:45pm**: progress report (at most 1 page). Any background papers you needed to read to better understand your core set. Thoughts on coherent and simple unifying presentation.

- **Monday, March 10th, 11:45pm**: updated short (≤ 1 page) writeup on more details of how you will present the ideas in a simple fashion.

- **Final presentations**: Monday, March 17, 2014, 2:30–4:20pm, LOW 102. What to turn in: your slides and a short at most 4 page summary of the papers.
As we saw, in a few examples, the expression for \( K(x) \) involved the entropy. How do they compare more generally?

First point: \( H \) requires a distribution while \( K \) does not (but then neither does Lempel-Ziv compression which, we can show, while algorithmic will converge to the entropy).

First, an important theorem.

**Theorem 29.2.3**

\[
\forall \mathcal{U} \text{ (meaning under any “reasonable” computing device),} \quad \sum_{p: \mathcal{U}(p) \text{ halts}} 2^{-\ell(p)} \leq 1 \tag{29.10}
\]

**Proof.**

If \( p \) halts, then it can’t be a prefix of any other program. Therefore, halting programs satisfy the prefix property, and we immediately have countably infinite Kraft inequality (from Lecture 9).
Bernoulli model of source

**Theorem 29.2.1**

Let $X_1, \ldots, X_n$ be $\sim$ Bernoulli$(1/2)$. Then

$$Pr(K(X_{1:n}|n) < n - k) < 2^{-k}$$  \hspace{1cm} (29.1)

**Proof.**

\[
Pr\left(K(X_{1:n}|n) < n - k\right) = \sum_{x_{1:n}:K(x_{1:n}|n) < n-k} Pr(x_{1:n})
\]

\[
= \sum_{x_{1:n}:K(x_{1:n}|n) < n-k} 2^{-n}
\]

\[
= \left|\{x_{1:n} : K(x_{1:n}|n) < n - k\}\right|2^{-n}
\]

\[
< 2^{n-k}2^{-n} = 2^{-k}
\]
Bernoulli model of source and Algorithmically Random

- Since $\Pr(K(X_1:n|n) < n - k) < 2^{-k}$, as we increase $k$ (i.e., compress $x_{1:n}$ by $k$ bits, from $n$ bits to $n - k$ bits), the probability of being able to do so decreases exponentially fast!
- Multiply the probabilities by $n$, and we get that the number of sequences of length $n$ that can be compressed by $k$ bits decreases exponentially fast in $k$!

**Definition 29.2.1**

$x_{1:n}$ is algorithmically random if

$$K(x_{1:n}) \geq n - c_n \quad \text{with } c_n \to 0 \quad \text{as } n \to \infty$$

So this is really a definition of a sequence of strings $\{x_{1:n}\}_n$ (or alternatively, an infinite length string that is considered at each point truncated to length $n$), such that, when $n$ gets big, the string becomes ever less compressible.

---

Prof. Jeff Bilmes
EE515a/Winter 2014/Information Theory II – Lecture 29 - Feb 12th, 2014

L29 F10/64 (pg.10/244)
Compressibility

Definition 29.2.1

A binary string $x_1, x_2, \ldots$, infinite in length, is algorithmically incompressible if:

$$\lim_{n \to \infty} \frac{K(x_1, x_2, \ldots, x_n \mid n)}{n} = 1$$

(29.1)

- So this is like $K(x_1:n \mid n) = n + c$.
- We know that if a random string is incompressible in an entropic sense, there are likely to be about half 0’s and half 1’s (i.e., the probability is close to 1/2). Is there an analogous notion for Kolmogorov complexity?
Compressibility

Theorem 29.2.1 (strong law)

If $x_{1:\infty}$ is incompressible, then

$$\lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} x_i = 1/2$$

(29.1)

Like a series of i.i.d. variables where $p(X_i = 1) = 1/2$, but note that we’ve said nothing about probabilities, rather we’ve only mentioned its algorithmic complexity.

Proof.

- Define $\theta_n = \frac{1}{n} \sum_{i=1}^{n} x_i$, then from Theorem 29.2.2,

$$n - c_n \leq K(x_{1:n}|n) \leq nH(\theta_n) + 2 \log n + c'$$

(29.2)

with $c_n/n \to 0$. Therefore,

$$H(\theta_n) \geq 1 - \frac{2 \log n + c_n + c'}{n}$$

(29.3)
Expected Per-symbol Kolmogorov Complexity

**Theorem 29.2.1**

Let $X_1, X_2, \ldots, X_n$ be i.i.d. Bernoulli($\theta$). Then

$$\frac{1}{n} K(X_1:n | n) \to H(\theta) \text{ in probability}$$

meaning $\Pr\left(|\frac{1}{n} K - H| > \epsilon\right) \to 0$ $\forall \epsilon > 0$

We’ll show:

- **A:** $\Pr(K > n(H + \epsilon)) \to 0$ as $n \to \infty$ for any $\epsilon > 0$, and
- **B:** $\Pr(K < n(H - c)) \to 0$ as $n \to \infty$ for any $c > 0$ ($c$ is used as an $\epsilon$ like quantity but we use symbol $c$ since we’ll also use $\epsilon$ in the same context below).
Universal Probability

- Suppose $H = 1$, so that the number of typical sequences of length $n$ is $2^{nH} = 2^n$, so all sequences are equally likely.
- Thus, $p(0, 0, \ldots, 0) = \frac{1}{2^L}$ where we have $L$ 0’s.
- Also, $p(1011011011101\ldots01101) = \frac{1}{2^L}$.
- Both the “all zeros” case, and the seemingly “random” sequence case are in the typical set and are thus considered equally likely.
- Which one is more surprising to you intuitively? Which one is more simple? Which one is easier to describe?
- Is Shannon’s theory lacking? It only gives complexity to an ensemble of messages (or a diversity of messages, not to the message contents). Shannon’s approach does not ask: How complex is the message $x_{1:n}$ itself?
- Shannon says that the “complexity” of the single message is the coding length $\log \frac{1}{p(x_{1:n})}$, irregardless of $x_{1:n}$.
- But how can we get to coding length without probabilities?
An Appeal to Philosophy: Occam’s Razor

- Occam’s razor: “If there are multiple explanations for a phenomenon, then all other things being equal, we should select the simplest one”
- “simplicity of an object” is the same as “the object has the shortest effective description”
- Occam’s razor says that the short explanations, or causes, of objects are the more probable ones.
- This is really a prior probability on nature itself.
Prefix-free programs: Occam’s razor prior probability

- Can we derive a probability distribution for this?
- We can quantify this by defining a probability on random programs that are prefix-free (halting). That is, for program \( p \), define:

\[
\Pr(p) = 2^{-\ell(p)}
\]  (29.14)

so that \( \sum_{p} \Pr(p) \leq 1 \) by Kraft inequality.
- This is an assumed fundamental property of nature and the laws of physics.
- Short programs are much more probable than long ones.
- Most sequences \( x_{1:n} \) of length \( n \) are complex (since there are just not enough short programs to go around to cover all the long sequences, of which there are many).
- When short programs produce long strings, they produce simple ones (by definition).
We can use this to produce a probability distribution on strings that is **universal**, where simple strings are more probable than complex ones of the same length.

**Definition 29.2.1**

The universal probability of a string $x$ is given by:

$$P_U(x) = \sum_{p : U(p) = x} 2^{-\ell(p)} = \Pr(U(\mathcal{P}) = x)$$ (29.14)

where on the r.h.s. $\mathcal{P}$ is a random program variable under $\Pr(\mathcal{P} = p)$.

This is also equal to the probability that random fair coin flips will create a program to print out $x$. 
The Meaning Of Life?

- $x$ could be natural images, natural scenes, or even sentience.
- And $p$ the program is what causes $x$ to occur.
- If nature acts as a grand computer in response to some random input, and if short programs are more probable than long ones, then this could cause $x$ to have structure even if $x$ is very big itself.
- Fractals images a good example, and often mimic real-world images.

while spectacular (e.g., NOVA http://youtu.be/s65DSz78jW4), fractals might be mundane vis-à-vis real world objects and their “programs.”
Halting Theorem

Theorem 29.2.1

There is no program $h(p, y)$ that takes a program $p$ with input $y$, where $h(p, y) = 1$ if $p$ halts on input $y$, and $h(p, y) = 0$ otherwise.

Proof.

- Suppose, to the contrary, that $\exists$ such an $h$, a halting indicator,
- and given $h$ we can easily construct program $\Psi(x)$ as follows:

  $\Psi(x) = \begin{cases} 
  1 & \text{if } h(x, x) = 0 \quad \text{i.e., } x \text{ does not halt on } x. \\
  \text{spin forever} & \text{if } h(x, x) = 1 \quad \text{i.e., } x \text{ halts on } x.
  \end{cases}$

(29.15)

$\Psi(x)$ asks: does $x$ halt on itself as input? If yes, we spin forever. I.e., $\Psi(x)$ spins forever if $x$ halts on itself, and halts if $x$ spins forever on itself.
Computing Kolmogorov Complexity

- Thus, $K_U(x)$ is not computable, since we cannot solve the halting problem.
- Suppose, just for fun, that we could compute $\Omega$.

$$0 \leq \Omega \triangleq \sum_{p:U(p) \text{ halts}} 2^{-\ell(p)} \leq 1$$ (29.16)

again which follows from the Kraft inequality.

- Key point: $0 < \Omega \leq 1$ actually exists (it is merely a number between 0 and 1) but we can’t compute it due to the halting theorem (this is actually an instance of Gödel’s theorem).
- But what if we could compute $\Omega$? What would this number tell us?
- Note, $\Omega$ is the halting probability, the probability that a program (drawn from the universal probability of programs) will halt.
Compressibility of $\Omega$

Even if we could compute $\Omega$, it is incompressible.

**Theorem 29.2.1**

$\Omega$ cannot be compressed by more than a constant. I.e., there exists a constant $c$ s.t.

$$K(\Omega_n) = K(\omega_1\omega_2\ldots\omega_n) \geq n - c \quad \forall n \quad (29.17)$$

**Proof.**

- Given $\Omega_n$, we can solve the halting problem for any program of length $\leq n$ bits.
- We can compute $\Omega_n$ with a length $K(\omega_1\omega_2\ldots\omega_n) = K(\Omega_n)$-bit program.
- Generate list of all programs length $\leq n$ that halt, and their output:

$$\left( p_{h_1}, x_{h_1} \right), \left( p_{h_2}, x_{h_2} \right), \ldots, \left( p_{h_m}, x_{h_m} \right) \quad (29.18)$$
We have a bound on the universal probability of a string $p_U(x)$ as follows:

$$p_U(x) = \sum_{p: U(p) = x} 2^{-\ell(p)} \geq 2^{-\ell(p^*)} = 2^{-K(x)} \quad (29.1)$$

where $p^* = \arg\min_{p: U(p) = x} \ell(p)$.
Universality

- We have a bound on the universal probability of a string $p_U(x)$ as follows:

$$p_U(x) = \sum_{p: U(p) = x} 2^{-\ell(p)} \geq 2^{-\ell(p^*)} = 2^{-K(x)} \quad (29.1)$$

where $p^* = \operatorname{argmin}_{p: U(p) = x} \ell(p)$

- In fact, we have that:

$$2^{-K(x)} \leq p_U(x) \leq c2^{-K(x)} \quad (29.2)$$
Universality

- We have a bound on the universal probability of a string \( p_U(x) \) as follows:

\[
p_U(x) = \sum_{p:U(p)=x} 2^{-\ell(p)} \geq 2^{-\ell(p^*)} = 2^{-K(x)} \quad (29.1)
\]

where \( p^* = \arg\min_{p:U(p)=x} \ell(p) \)

- In fact, we have that:

\[
2^{-K(x)} \leq p_U(x) \leq c2^{-K(x)} \quad (29.2)
\]

- or

\[
K(x) - c' \leq \log \frac{1}{p_U(x)} \leq K(x) \quad (29.3)
\]

so that \( \frac{1}{p_U(x)} \) is the Shannon “information” of \( x \).
We have a bound on the universal probability of a string $p_U(x)$ as follows:

$$p_U(x) = \sum_{p:U(p)=x} 2^{-\ell(p)} \geq 2^{-\ell(p^*)} = 2^{-K(x)} \quad (29.1)$$

where $p^* = \arg\min_{p:U(p)=x} \ell(p)$

In fact, we have that:

$$2^{-K(x)} \leq p_U(x) \leq c2^{-K(x)} \quad (29.2)$$

or

$$K(x) - c' \leq \log \frac{1}{p_U(x)} \leq K(x) \quad (29.3)$$

so that $\frac{1}{p_U(x)}$ is the Shannon “information” of $x$.

So, Shannon lengths are as natural as $K(x)$ when using $p_U(x)$ as a distribution.
Also, the entropy of \( p_U(x) \) is \( EK(X) \) where \( X \) is a random variable drawn via \( p_U \).
Complexity of the Universe

- Also, the entropy of $p_U(x)$ is $E K(X)$ where $X$ is a random variable drawn via $p_U$.

- So, we can think of this as the universal entropy or the universal complexity of the universe.
Also, the entropy of $p_U(x)$ is $EK(X)$ where $X$ is a random variable drawn via $p_U$.

So, we can think of this as the universal entropy or the universal complexity of the universe.

All, assuming, the Occam’s razor assumption as being the guiding principle of the universe.
The complexity of a string $x$ is the length of the shortest program that can generate $x$, when run on a universal computer $\mathcal{U}$.
Kolmogorov Complexity: Review so far

- The complexity of a string $x$ is the length of the shortest program that can generate $x$, when run on a universal computer $\mathcal{U}$.
- This is an algorithmic definition, and strings whose shortest program length are as long as the string itself are called algorithmically random (or algorithmically incompressible).
Kolmogorov Complexity: Review so far

- The complexity of a string $x$ is the length of the shortest program that can generate $x$, when run on a universal computer $U$.
- This is an algorithmic definition, and strings whose shortest program length are as long as the string itself are called algorithmically random (or algorithmically incompressible).
- We may think of structure in the universe as arising this way — Occam’s razor dictates (or guides us to decide) that short halting programs should be more probable, and a natural probability for halting program $p$ is $\Pr(p) = 2^{-\ell(p)}$
Kolmogorov Complexity: Review so far

- The complexity of a string $x$ is the length of the shortest program that can generate $x$, when run on a universal computer $\mathcal{U}$.
- This is an algorithmic definition, and strings whose shortest program length are as long as the string itself are called algorithmically random (or algorithmically incompressible).
- We may think of structure in the universe as arising this way — Occam’s razor dictates (or guides us to decide) that short halting programs should be more probable, and a natural probability for halting program $p$ is $\Pr(p) = 2^{-\ell(p)}$
- This gives us the universal probability $P_{\mathcal{U}}(x)$ of string $x$, but which is not computable (halting problem).
Kolmogorov Complexity: Review so far

- The complexity of a string $x$ is the length of the shortest program that can generate $x$, when run on a universal computer $\mathcal{U}$.
- This is an algorithmic definition, and strings whose shortest program length are as long as the string itself are called algorithmically random (or algorithmically incompressible).
- We may think of structure in the universe as arising this way — Occam’s razor dictates (or guides us to decide) that short halting programs should be more probable, and a natural probability for halting program $p$ is $\Pr(p) = 2^{-\ell(p)}$
- This gives us the universal probability $P_\mathcal{U}(x)$ of string $x$, but which is not computable (halting problem).
- $0 \leq \Omega \leq 1$, if available, would be an oracle. While we can’t compute $\Omega$, we know it exists, and we can deduce that it is (unsurprisingly, since it is an oracle) incompressible.
Kolmogorov Complexity: Review so far

- The complexity of a string $x$ is the length of the shortest program that can generate $x$, when run on a universal computer $\mathcal{U}$.
- This is an algorithmic definition, and strings whose shortest program length are as long as the string itself are called algorithmically random (or algorithmically incompressible).
- We may think of structure in the universe as arising this way — Occam’s razor dictates (or guides us to decide) that short halting programs should be more probable, and a natural probability for halting program $p$ is $\Pr(p) = 2^{-\ell(p)}$
- This gives us the universal probability $P_{\mathcal{U}}(x)$ of string $x$, but which is not computable (halting problem).
- $0 \leq \Omega \leq 1$, if available, would be an oracle. While we can’t compute $\Omega$, we know it exists, and we can deduce that it is (unsurprisingly, since it is an oracle) incompressible.
- Even though $K(x)$ is algorithmic, we saw several ways it relates to (and sometimes is) entropy.
While $K(x)$ is algorithmic, and sometimes relates to entropy, it is not a practical measure since we can’t compute it.
While $K(x)$ is algorithmic, and sometimes relates to entropy, it is not a practical measure since we can’t compute it.

Are there any purely algorithmic forms of compression (besides $K$) that can be shown to relate to $H$ and, ideally, can be shown to compress to the entropy rate?
While $K(x)$ is algorithmic, and sometimes relates to entropy, it is not a practical measure since we can’t compute it.

Are there any purely algorithmic forms of compression (besides $K$) that can be shown to relate to $H$ and, ideally, can be shown to compress to the entropy rate?

Note, that if it is algorithmic, it would be useful if it doesn’t need to explicitly compute the probability distribution governing the symbols.
While $K(x)$ is algorithmic, and sometimes relates to entropy, it is not a practical measure since we can’t compute it.

Are there any purely algorithmic forms of compression (besides $K$) that can be shown to relate to $H$ and, ideally, can be shown to compress to the entropy rate?

Note, that if it is algorithmic, it would be useful if it doesn’t need to explicitly compute the probability distribution governing the symbols.

I.e., do there exist compression algorithms that do not use the probability distribution but still give the entropy rate?
Recall: Fixed Length Block Codes

- We had fixed number of source symbols, fixed code length (fixed length codewords)
Recall: Fixed Length Block Codes

We had fixed number of source symbols, fixed code length (fixed length codewords)

ex: AEP, and the method of types
Recall: Fixed Length Block Codes

- We had fixed number of source symbols, fixed code length (fixed length codewords)
- ex: AEP, and the method of types
- Only a small number of source sequences gave code words
Recall: Fixed Length Block Codes

- We had fixed number of source symbols, fixed code length (fixed length codewords)
- ex: AEP, and the method of types
- Only a small number of source sequences gave code words
- Good for entropy and proofs of existence of codes, but not very practical
Recall: Symbol Codes

- Variable length codewords for each source symbol.
Recall: Symbol Codes

- Variable length codewords for each source symbol.
- More probable symbols gave shorter length encodings.
Recall: Symbol Codes

- Variable length codewords for each source symbol.
- More probable symbols gave shorter length encodings.
- Ex: Huffman codes
Recall: Symbol Codes

- Variable length codewords for each source symbol.
- More probable symbols gave shorter length encodings.
- Ex: Huffman codes
- Need distribution, and penalty if a mismatch of $D(p||q)$
Recall: Symbol Codes

- Variable length codewords for each source symbol.
- More probable symbols gave shorter length encodings.
- Ex: Huffman codes
- Need distribution, and penalty if a mismatch of $D(p||q)$
- Still requires blocking of the symbols in order to achieve the entropy rate (which occurs in the limit).
Recall: Stream Codes

- Here, we do not necessarily emit bits for every source symbol, might need to wait and then emit bits for a sequence which could be variable length.
Recall: Stream Codes

- Here, we do not necessarily emit bits for every source symbol, might need to wait and then emit bits for a sequence which could be variable length.

- Large number of source symbols can be represented with small number of bits without fixed blocking.

Ex: Arithmetic codes (uses a subinterval of $[0, 1]$ corresponding to $p(x_1, ..., x_{n-1})$) and requires a model (ideally adaptive) of the source sequence.

Ex: adaptive arithmetic codes (Dirichlet distribution).

Next example we will cover in class is Lempel-Ziv. Basic idea, memorize strings that have already occurred, without even modeling source distribution.
Recall: Stream Codes

- Here, we do not necessarily emit bits for every source symbol, might need to wait and then emit bits for a sequence which could be variable length.
- Large number of source symbols can be represented with small number of bits without fixed blocking.
- Can start encoding quickly, and can start decoding before encoding is even finished.
Recall: Stream Codes

- Here, we do not necessarily emit bits for every source symbol, might need to wait and then emit bits for a sequence which could be variable length.
- Large number of source symbols can be represented with small number of bits without fixed blocking.
- Can start encoding quickly, and can start decoding before encoding is even finished.
- **Ex: Arithmetic codes** (uses a subinterval of $[0, 1]$ corresponding to $p(x_n|x_1, \ldots, x_{n-1})$) and requires a model (ideally adaptive) of the source sequence.
Recall: Stream Codes

- Here, we do not necessarily emit bits for every source symbol, might need to wait and then emit bits for a sequence which could be variable length.
- Large number of source symbols can be represented with small number of bits without fixed blocking.
- Can start encoding quickly, and can start decoding before encoding is even finished.
- Ex: Arithmetic codes (uses a subinterval of $[0, 1]$ corresponding to $p(x_n|x_1, \ldots, x_{n-1})$) and requires a model (ideally adaptive) of the source sequence.
- Ex: adaptive arithmetic codes (Dirichlet distribution).
Recall: Stream Codes

- Here, we do not necessarily emit bits for every source symbol, might need to wait and then emit bits for a sequence which could be variable length.
- Large number of source symbols can be represented with small number of bits without fixed blocking.
- Can start encoding quickly, and can start decoding before encoding is even finished.
- Ex: Arithmetic codes (uses a subinterval of \([0, 1]\) corresponding to \(p(x_n|x_1, \ldots, x_{n-1})\)) and requires a model (ideally adaptive) of the source sequence.
- Ex: adaptive arithmetic codes (Dirichlet distribution).
- Next example we will cover in class is Lempel-Ziv. Basic idea, memorize strings that have already occurred, without even modeling source distribution.
Huffman is inherently 2-pass. We use 1st pass to estimate $p(x)$, but this might not be feasible (we might want to start compressing right away, which is the reason for stream codes).
Universal Source Coding

- Huffman is inherently 2-pass. We use 1st pass to estimate $p(x)$, but this might not be feasible (we might want to start compressing right away, which is the reason for stream codes).
- Huffman pays at most one extra bit per symbol. To achieve entropy rate might need a long block length, where jointly encoding independent source symbols is beneficial in terms of average number of bits per symbol in length—$n$ block: $\frac{nH(X)+1}{n}$. 
Universal Source Coding

- Huffman is inherently 2-pass. We use 1st pass to estimate $p(x)$, but this might not be feasible (we might want to start compressing right away, which is the reason for stream codes).

- Huffman pays at most one extra bit per symbol. To achieve entropy rate might need a long block length, where jointly encoding independent source symbols is beneficial in terms of average number of bits per symbol in length $n$ block: $\frac{nH(X)+1}{n}$.

- In general, we want to be able to code down to the entropy without needing the source distribution (explicitly).
Universal Source Coding

- Huffman is inherently 2-pass. We use 1st pass to estimate $p(x)$, but this might not be feasible (we might want to start compressing right away, which is the reason for stream codes).

- Huffman pays at most one extra bit per symbol. To achieve entropy rate might need a long block length, where jointly encoding independent source symbols is beneficial in terms of average number of bits per symbol in length $n$ block: $\frac{nH(X) + 1}{n}$.

- In general, we want to be able to code down to the entropy without needing the source distribution (explicitly).

- Want universal in the sense that if $p(x)$ remains unknown, we can still code at $H(X)$ bits per symbol.
Universal Source Coding

- Huffman is inherently 2-pass. We use 1st pass to estimate $p(x)$, but this might not be feasible (we might want to start compressing right away, which is the reason for stream codes).
- Huffman pays at most one extra bit per symbol. To achieve entropy rate might need a long block length, where jointly encoding independent source symbols is beneficial in terms of average number of bits per symbol in length $- n$ block: $\frac{nH(X)+1}{n}$.
- In general, we want to be able to code down to the entropy without needing the source distribution (explicitly).
- Want universal in the sense that if $p(x)$ remains unknown, we can still code at $H(X)$ bits per symbol.
- Lempel-Ziv coding is universal (as we will see).
Universal Source Coding

- Huffman is inherently 2-pass. We use 1st pass to estimate $p(x)$, but this might not be feasible (we might want to start compressing right away, which is the reason for stream codes).
- Huffman pays at most one extra bit per symbol. To achieve entropy rate might need a long block length, where jointly encoding independent source symbols is beneficial in terms of average number of bits per symbol in length $n$ block: $\frac{nH(X)+1}{n}$.
- In general, we want to be able to code down to the entropy without needing the source distribution (explicitly).
- Want universal in the sense that if $p(x)$ remains unknown, we can still code at $H(X)$ bits per symbol.
- Lempel-Ziv coding is universal (as we will see).
- It is also the algorithm used in gzip, the widely used text compression algorithm (although bzip2 often compresses a bit better, which uses the Burrows-Wheeler block-sorting text compression algorithm along with Huffman coding).
The source sequence is parsed into shortest “phrases” that have not yet been seen so far (in the previous history of phrases).
Lempel Ziv Compression

- The source sequence is parsed into shortest “phrases” that have not yet been seen so far (in the previous history of phrases).
- **Ex:** for the string “this is the thing”, we have the following parse:
Lempel Ziv Compression

- The source sequence is parsed into shortest "phrases" that have not yet been seen so far (in the previous history of phrases).
- Ex: for the string "this is the thing", we have the following parse:
  
  \[ t, \]
Lempel Ziv Compression

- The source sequence is parsed into shortest “phrases” that have not yet been seen so far (in the previous history of phrases).
- Ex: for the string “this is the thing”, we have the following parse:
  
  $$t, h,$$
Lempel Ziv Compression

- The source sequence is parsed into shortest “phrases” that have not yet been seen so far (in the previous history of phrases).
- Ex: for the string “this is the thing”, we have the following parse: \( t, h, i \).
The source sequence is parsed into shortest “phrases” that have not yet been seen so far (in the previous history of phrases).

Ex: for the string “this is the thing”, we have the following parse:

\[ t, h, i, s, \]
Lempel Ziv Compression

- The source sequence is parsed into shortest “phrases” that have not yet been seen so far (in the previous history of phrases).
- Ex: for the string “this is the thing”, we have the following parse:
  
  \[ t, h, i, s, \ldots \]
Lempel Ziv Compression

- The source sequence is parsed into shortest “phrases” that have not yet been seen so far (in the previous history of phrases).
- Ex: for the string “this is the thing”, we have the following parse:
  
  $t, \ h, \ i, \ s, \ \omega, \ is,$
Lempel Ziv Compression

- The source sequence is parsed into shortest “phrases” that have not yet been seen so far (in the previous history of phrases).
- Ex: for the string “this is the thing”, we have the following parse:
  \[ t, h, i, s, \text{-}, is, \text{-}t, \]
Lempel Ziv Compression

- The source sequence is parsed into shortest “phrases” that have not yet been seen so far (in the previous history of phrases).
- Ex: for the string “this is the thing”, we have the following parse:
  
  \[ t, h, i, s, \omega, is, \omega t, he, \]
Lempel Ziv Compression

- The source sequence is parsed into shortest “phrases” that have not yet been seen so far (in the previous history of phrases).
- Ex: for the string “this is the thing”, we have the following parse:
  
  \[ t, h, i, s, \_, is, \_, t, he, \_th, \]
Lempel Ziv Compression

- The source sequence is parsed into shortest “phrases” that have not yet been seen so far (in the previous history of phrases).
- Ex: for the string “this is the thing”, we have the following parse:
  
  \[ t, h, i, s, \land, is, \land t, he, \land th, in, \]
Lempel Ziv Compression

- The source sequence is parsed into shortest “phrases” that have not yet been seen so far (in the previous history of phrases).
- Ex: for the string “this is the thing”, we have the following parse:
  \[ t, h, i, s, \_, is, \_, t, he, \_, th, in, g \]
The source sequence is parsed into shortest “phrases” that have not yet been seen so far (in the previous history of phrases).

Ex: for the string “this is the thing”, we have the following parse:

\[ t, h, i, s, is, t, he, th, in, g \]

Ex: Binary string “1011010100010 . . . ” gets parsed as
Lempel Ziv Compression

- The source sequence is parsed into shortest “phrases” that have not yet been seen so far (in the previous history of phrases).
- Ex: for the string “this is the thing”, we have the following parse:
  
  \[ t, h, i, s, \, \_\, \, is, \, \_t, \, he, \, \_\, th, \, in, \, g \]

- Ex: Binary string “1011010100010 …” gets parsed as
  
  \[ 1, \, 0, \, 1, \, 1, \, 0, \, 1, \, 0, \, 1, \, 0, \, 0, \, 1, \, 0, \, \ldots \]
The source sequence is parsed into shortest “phrases” that have not yet been seen so far (in the previous history of phrases).

Ex: for the string “this is the thing”, we have the following parse:

\[ t, h, i, s, \text{is}, \text{it}, h, e, \text{th}, i, n, g \]

Ex: Binary string “1011010100010 …” gets parsed as

\[ 1, 0, 11, 01, 010, 00, 10, \ldots \]

So for each new part of the string, we look for the shortest string not seen so far, in the previous list of phrases.
Lempel Ziv Compression

- The source sequence is parsed into shortest “phrases” that have not yet been seen so far (in the previous history of phrases).
- Ex: for the string “this is the thing”, we have the following parse:
  
  $t, h, i, s, \ldots, is, \ldots, t, he, \ldots, th, in, g$

- Ex: Binary string “1011010100010 \ldots” gets parsed as
  
  $1, 0, 11, 01, 010, 00, 10, \ldots$

- So for each new part of the string, we look for the shortest string not seen so far, in the previous list of phrases.
- Thus, all prefixes of any token occurred before, up to the very last symbol (or bit in the binary case).
Lempel Ziv Encoding

To encode, we give the location of the prefix (which is everything but the final symbol) and then append that index with the final symbol. Use 0 as a null pointer, indicating the string didn’t occur before.
Lempel Ziv Encoding

- To encode, we give the location of the prefix (which is everything but the final symbol) and then append that index with the final symbol. Use 0 as a null pointer, indicating the string didn’t occur before.
- Ex: for the string “this is the thing”, we have the following parse and encoding:

<table>
<thead>
<tr>
<th>phrase</th>
<th>t</th>
<th>h</th>
<th>i</th>
<th>s</th>
<th></th>
<th>is</th>
<th></th>
<th>t</th>
<th>he</th>
<th></th>
<th>th</th>
<th>in</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>encoding</td>
<td>(0,t)</td>
<td>(0,h)</td>
<td>(0,i)</td>
<td>(0,s)</td>
<td>(0,\omega)</td>
<td>(3,s)</td>
<td>(5,t)</td>
<td>(2,e)</td>
<td>(7,h)</td>
<td>(3,n)</td>
<td>(0,g)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Lempel Ziv Encoding

- To encode, we give the location of the prefix (which is everything but the final symbol) and then append that index with the final symbol. Use 0 as a null pointer, indicating the string didn’t occur before.
- Ex: for the string “this is the thing”, we have the following parse and encoding:

<table>
<thead>
<tr>
<th>phrase</th>
<th>t</th>
<th>h</th>
<th>i</th>
<th>s</th>
<th></th>
<th>is</th>
<th></th>
<th>t</th>
<th>he</th>
<th></th>
<th>th</th>
<th>in</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>encoding</td>
<td>(0,t)</td>
<td>(0,h)</td>
<td>(0,i)</td>
<td>(0,s)</td>
<td>(0,)</td>
<td>(3,s)</td>
<td>(5,t)</td>
<td>(2,e)</td>
<td>(7,h)</td>
<td>(3,n)</td>
<td>(0,g)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Ex: Binary string “1011010100010 …”, we have the following parse

<table>
<thead>
<tr>
<th>phrase</th>
<th>1</th>
<th>0</th>
<th>11</th>
<th>01</th>
<th>010</th>
<th>00</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>encoding</td>
<td>(0,1)</td>
<td>(0,0)</td>
<td>(1,1)</td>
<td>(2,1)</td>
<td>(4,0)</td>
<td>(2,0)</td>
<td>(1,0)</td>
</tr>
</tbody>
</table>
Lempel Ziv Encoding

- To encode, we give the location of the prefix (which is everything but the final symbol) and then append that index with the final symbol. Use 0 as a null pointer, indicating the string didn’t occur before.
- Ex: for the string “this is the thing”, we have the following parse and encoding:

<table>
<thead>
<tr>
<th>phrase</th>
<th>t</th>
<th>h</th>
<th>i</th>
<th>s</th>
<th></th>
<th>is</th>
<th></th>
<th>t</th>
<th>he</th>
<th></th>
<th>th</th>
<th>in</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>encoding</td>
<td>(0,t)</td>
<td>(0,h)</td>
<td>(0,i)</td>
<td>(0,s)</td>
<td>(0,,,,)</td>
<td>(3,s)</td>
<td>(5,t)</td>
<td>(2,e)</td>
<td>(7,h)</td>
<td>(3,n)</td>
<td>(0,g)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Ex: Binary string “1011010100010 . . . ”, we have the following parse

<table>
<thead>
<tr>
<th>phrase</th>
<th>1</th>
<th>0</th>
<th>11</th>
<th>01</th>
<th>010</th>
<th>00</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>encoding</td>
<td>(0,1)</td>
<td>(0,0)</td>
<td>(1,1)</td>
<td>(2,1)</td>
<td>(4,0)</td>
<td>(2,0)</td>
<td>(1,0)</td>
</tr>
</tbody>
</table>

- In general, as encoding proceeds, pointer refers back to longer and longer strings, and compression rate improves, assuming that there is statistical regularly (and repeated strings) in the source.
Let \( c(n) \) be the number of phrases of a string of length \( n \) parsed in this way (shortest string not seen so far).
Lempel Ziv: Binary Encoding

- Let $c(n)$ be the number of phrases of a string of length $n$ parsed in this way (shortest string not seen so far).
- Need $\log c(n)$ bits to include a pointer to all previous locations,
Lempel Ziv: Binary Encoding

- Let $c(n)$ be the number of phrases of a string of length $n$ parsed in this way (shortest string not seen so far).
- Need $\log c(n)$ bits to include a pointer to all previous locations, +1 additional bit needed to describe the final bit.
Lempel Ziv: Binary Encoding

- Let \( c(n) \) be the number of phrases of a string of length \( n \) parsed in this way (shortest string not seen so far).
- Need \( \log c(n) \) bits to include a pointer to all previous locations, +1 additional bit needed to describe the final bit.
- Ex: \( c(n) = 7 \). We have \( (000, 1) \) \( (000, 0) \) \( (001, 1) \) \( (010, 1) \) \( (100, 0) \) \( (010, 0) \) \( (001, 0) \)
Lempel Ziv: Binary Encoding

- Let \(c(n)\) be the number of phrases of a string of length \(n\) parsed in this way (shortest string not seen so far).
- Need \(\log c(n)\) bits to include a pointer to all previous locations, +1 additional bit needed to describe the final bit.
- Ex: \(c(n) = 7\). We have \((000, 1), (000, 0), (001, 1), (010, 1), (100, 0)\)
  \((010, 0), (001, 0)\)
- Decoding is easy: just keep strings we have constructed already, and when we encounter \((i, j)\) just output string \(i\) and follow it with \(j\).
Lempel Ziv: Binary Encoding

- Let $c(n)$ be the number of phrases of a string of length $n$ parsed in this way (shortest string not seen so far).
- Need $\log c(n)$ bits to include a pointer to all previous locations, +1 additional bit needed to describe the final bit.
- Ex: $c(n) = 7$. We have $(000, 1)$ (000, 0) (001, 1) (010, 1) (100, 0)

  $(010, 0)$ (001, 0)

  1 2 3 4 5

  6 7

- Decoding is easy: just keep strings we have constructed already, and when we encounter $(i, j)$ just output string $i$ and follow it with $j$.
- When might this work well with low-entropy source?
Lempel Ziv: Binary Encoding

- Let \( c(n) \) be the number of phrases of a string of length \( n \) parsed in this way (shortest string not seen so far).
- Need \( \log c(n) \) bits to include a pointer to all previous locations, +1 additional bit needed to describe the final bit.
- Ex: \( c(n) = 7 \). We have \((000, 1) (000, 0) (001, 1) (010, 1) (100, 0) \)

\[(010, 0) (001, 0) \]

6 7

- Decoding is easy: just keep strings we have constructed already, and when we encounter \((i, j)\) just output string \( i \) and follow it with \( j \).
- When might this work well with low-entropy source? Lots of repeated strings and sub-strings, e.g., real-world signals such as text.
Lempel Ziv: Binary Encoding

- Let $c(n)$ be the number of phrases of a string of length $n$ parsed in this way (shortest string not seen so far).
- Need $\log c(n)$ bits to include a pointer to all previous locations, +1 additional bit needed to describe the final bit.
- Ex: $c(n) = 7$. We have $(000, 1)$ (000, 0) (001, 1) (010, 1) (100, 0)

$(010, 0)$ (001, 0)

$(010, 0)$ (001, 0)

Decoding is easy: just keep strings we have constructed already, and when we encounter $(i, j)$ just output string $i$ and follow it with $j$.
- When might this work well with low-entropy source? Lots of repeated strings and sub-strings, e.g., real-world signals such as text.
- When might this fail with a low-entropy source?
Lempel Ziv: Binary Encoding

- Let $c(n)$ be the number of phrases of a string of length $n$ parsed in this way (shortest string not seen so far).
- Need $\log c(n)$ bits to include a pointer to all previous locations, +1 additional bit needed to describe the final bit.
- Ex: $c(n) = 7$. We have $(000, 1)$, $(000, 0)$, $(001, 1)$, $(010, 1)$, $(100, 0)$, $(010, 0)$, $(001, 0)$.
- Decoding is easy: just keep strings we have constructed already, and when we encounter $(i, j)$ just output string $i$ and follow it with $j$.
- When might this work well with low-entropy source? Lots of repeated strings and sub-strings, e.g., real-world signals such as text.
- When might this fail with a low-entropy source? Lots of long-range and sparse correlations, or lots of almost repeats (e.g., sampled continuous signals, where the effective alphabet is large, e.g., speech, music, sound, image).
Lempel-Ziv

- Assume source alphabet is binary ($\mathcal{X} = \{0, 1\}$).
Lempel-Ziv

- Assume source alphabet is binary ($\mathcal{X} = \{0, 1\}$).
- Define:
Lempel-Ziv

- Assume source alphabet is binary ($\mathcal{X} = \{0, 1\}$).
- Define:

**Definition 29.5.1**

A parsing $S$ of a binary string $x_1, x_2, \ldots, x_n$ is a division of the string into phrases, separated by commas.
Assume source alphabet is binary ($\mathcal{X} = \{0, 1\}$).

Define:

**Definition 29.5.1**

A parsing $S$ of a binary string $x_1, x_2, \ldots, x_n$ is a division of the string into phrases, separated by commas.

**Definition 29.5.2**

A **distinct** parsing is a parsing where no two phrases are identical.
Assume source alphabet is binary ($\mathcal{X} = \{0, 1\}$).

Define:

**Definition 29.5.1**
A parsing $S$ of a binary string $x_1, x_2, \ldots, x_n$ is a division of the string into phrases, separated by commas.

**Definition 29.5.2**
A **distinct** parsing is a parsing where no two phrases are identical.

- Ex: 01101101. A parsing is 0,11,0,11,01 but this is not distinct.
Lempel-Ziv

- Assume source alphabet is binary ($\mathcal{X} = \{0, 1\}$).

- Define:

**Definition 29.5.1**

A parsing $S$ of a binary string $x_1, x_2, \ldots, x_n$ is a division of the string into phrases, separated by commas.

**Definition 29.5.2**

A distinct parsing is a parsing where no two phrases are identical.

- Ex: 01101101. A parsing is $0, 11, 0, 11, 01$ but this is not distinct.

- A distinct parsing of 01101101 would be $0, 1, 10, 11, 01$
Lempel-Ziv

- Assume source alphabet is binary \( \mathcal{X} = \{0, 1\} \).
- Define:

**Definition 29.5.1**

A parsing \( S \) of a binary string \( x_1, x_2, \ldots, x_n \) is a division of the string into phrases, separated by commas.

**Definition 29.5.2**

A **distinct** parsing is a parsing where no two phrases are identical.

- Ex: 01101101. A parsing is 0,1,10,11,01 but this is not distinct.
- A distinct parsing of 01101101 would be 0,1,10,11,01
- Of course, Lempel-Ziv produces a distinct parsing of the source sequence.
Let $c(n)$ be the number of phrases in a LZ parsing of the string of length $n$. Thus, $c(n)$ depends on the string $x_{1:n}$.

$(c = c(n) = c(x_{1:n}))$. 

After compression, we have a sequence of $c(n)$ pairs of numbers of the form $(\text{pointer, bit})$ where each pointer requires $\lceil \log c(n) \rceil$ bits.

Length of the compressed sequence is then $c(n)(\log c(n) + 1)$ bits (29.4)

Can we show that LZ compression achieves the entropy rate? How?

I.e., our goal is to show that:

$c(n)(\log c(n) + 1) n \rightarrow H(X)$ (29.5)

for stationary ergodic sequence $x_{1}, x_{2}, ..., x_{n}$.
Lempel-Ziv

Let $c(n)$ be the number of phrases in a LZ parsing of the string of length $n$. Thus, $c(n)$ depends on the string $x_{1:n}$ ($c = c(n) = c(x_{1:n})$).

After compression, we have a sequence of $c(n)$ pairs of numbers of the form (pointer, bit) where each pointer requires $\lceil \log c(n) \rceil$ bits.
Lempel-Ziv

- Let $c(n)$ be the number of phrases in a LZ parsing of the string of length $n$. Thus, $c(n)$ depends on the string $x_{1:n}$ ($c = c(n) = c(x_{1:n})$).
- After compression, we have a sequence of $c(n)$ pairs of numbers of the form (pointer, bit) where each pointer requires $\lceil \log c(n) \rceil$ bits.
- Length of the compressed sequence is then

$$c(n) (\log c(n) + 1) \text{ bits} \quad (29.4)$$
Lempel-Ziv

- Let $c(n)$ be the number of phrases in a LZ parsing of the string of length $n$. Thus, $c(n)$ depends on the string $x_{1:n}$
  
  \[ c = c(n) = c(x_{1:n}) \].

- After compression, we have a sequence of $c(n)$ pairs of numbers of the form (pointer, bit) where each pointer requires $\lceil \log c(n) \rceil$ bits.

- Length of the compressed sequence is then

  \[ c(n)(\log c(n) + 1) \text{ bits} \quad (29.4) \]

- Can we show that LZ compression achieves the entropy rate? How?
Lempel-Ziv

- Let $c(n)$ be the number of phrases in a LZ parsing of the string of length $n$. Thus, $c(n)$ depends on the string $x_{1:n}$ ($c = c(n) = c(x_{1:n})$).

- After compression, we have a sequence of $c(n)$ pairs of numbers of the form (pointer, bit) where each pointer requires $\lceil \log c(n) \rceil$ bits.

- Length of the compressed sequence is then

$$c(n)(\log c(n) + 1) \text{ bits}$$

- Can we show that LZ compression achieves the entropy rate? How?

- I.e., our goal is to show that:

$$\frac{c(n)(\log c(n) + 1)}{n} \rightarrow H(X) \quad \text{(29.5)}$$

for stationary ergodic sequence $x_1, x_2, \ldots, x_n$. 
A little note on little "oh" $o(g(n))$ notation

- When we say $o(g(n))$, we mean:
  
  $$o(g(n)) \triangleq \{ f(n) : \forall c > 0, \exists n_0 > 0 \text{ s.t. } 0 \leq f(n) < cg(n), \forall n \geq n_0 \}$$
A little note on little “oh” $o(g(n))$ notation

When we say $o(g(n))$, we mean:

$$o(g(n)) \triangleq \{ f(n) : \forall c > 0, \exists n_0 > 0 \text{ s.t. } 0 \leq f(n) < cg(n), \forall n \geq n_0 \}$$

I.e., $o(g(n))$ is all functions that for any constant $c$ are eventually sandwiched above by $cg(n)$.
A little note on little “oh” $o(g(n))$ notation

- When we say $o(g(n))$, we mean:
  \[ o(g(n)) \triangleq \{ f(n) : \forall c > 0, \exists n_0 > 0 \text{ s.t. } 0 \leq f(n) < cg(n), \forall n \geq n_0 \} \]

  I.e., $o(g(n))$ is all functions that for any constant $c$ are eventually sandwiched above by $cg(n)$.

- $o(g(n))$ is a set of functions, each of which satisfies the constraint given above.
A little note on little “oh” $o(g(n))$ notation

- When we say $o(g(n))$, we mean:

  $$o(g(n)) \triangleq \{ f(n) : \forall c > 0, \exists n_0 > 0 \text{ s.t. } 0 \leq f(n) < cg(n), \forall n \geq n_0 \}$$

  I.e., $o(g(n))$ is all functions that for any constant $c$ are eventually sandwiched above by $cg(n)$.

- $o(g(n))$ is a set of functions, each of which satisfies the constraint given above.

- Intuitively, if $f \in o(g(n))$, it means that $g(n)$ grows faster than $f(n)$, eventually.
A little note on little “oh” $o(g(n))$ notation

- When we say $o(g(n))$, we mean:

$$o(g(n)) \triangleq \{ f(n) : \forall c > 0, \exists n_0 > 0 \text{ s.t. } 0 \leq f(n) < cg(n), \forall n \geq n_0 \}$$

I.e., $o(g(n))$ is all functions that for any constant $c$ are eventually sandwiched above by $cg(n)$.

- $o(g(n))$ is a set of functions, each of which satisfies the constraint given above.

- Intuitively, if $f \in o(g(n))$, it means that $g(n)$ grows faster than $f(n)$, eventually.

- If $g(n)$ is not zero, then for any $f \in o(g(n))$, it is like

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0.$$
A little note on little “oh” $o(g(n))$ notation

- When we say $o(g(n))$, we mean:

$$o(g(n)) \triangleq \{ f(n) : \forall c > 0, \exists n_0 > 0 \text{ s.t. } 0 \leq f(n) < cg(n), \forall n \geq n_0 \}$$

I.e., $o(g(n))$ is all functions that for any constant $c$ are eventually sandwiched above by $cg(n)$.

- $o(g(n))$ is a set of functions, each of which satisfies the constraint given above.

- Intuitively, if $f \in o(g(n))$, it means that $g(n)$ grows faster than $f(n)$, eventually.

- If $g(n)$ is not zero, then for any $f \in o(g(n))$, it is like

$$\lim_{n \to \infty} f(n)/g(n) = 0.$$  

- Then $o(1)$ are functions that go to zero in the limit, that is

$$o(1) \triangleq \{ f(n) : \forall c > 0, \exists n_0 > 0 \text{ s.t. } 0 \leq f(n) < c, \forall n \geq n_0 \}$$
When we say $o(g(n))$, we mean:

$$o(g(n)) \triangleq \{ f(n) : \forall c > 0, \exists n_0 > 0 \text{ s.t. } 0 \leq f(n) < cg(n), \forall n \geq n_0 \}$$

I.e., $o(g(n))$ is all functions that for any constant $c$ are eventually sandwiched above by $cg(n)$.

- $o(g(n))$ is a set of functions, each of which satisfies the constraint given above.
- Intuitively, if $f \in o(g(n))$, it means that $g(n)$ grows faster than $f(n)$, eventually.
- If $g(n)$ is not zero, then for any $f \in o(g(n))$, it is like
  $$\lim_{n \to \infty} f(n)/g(n) = 0.$$
- Then $o(1)$ are functions that go to zero in the limit, that is
  $$o(1) \triangleq \{ f(n) : \forall c > 0, \exists n_0 > 0 \text{ s.t. } 0 \leq f(n) < c, \forall n \geq n_0 \}$$
- Thus, if $f \in o(1)$, then
  $$\lim_{n \to \infty} f(n) = 0.$$
Lemma 29.5.3

The number of phrases $c(n)$ in any distinct parsing of binary sequence $x_1^n$ satisfies:

$$c(n) \leq \frac{n}{(1 - \epsilon_n) \log n}$$

where $\epsilon \to 0$ as $n \to \infty$. That is

$$c(n) \leq \frac{n}{\log n} \left(1 + o(1)\right)$$

Proof.

Let $n_k$ be the sum of lengths of all distinct strings of length $\leq k$. To express $n_k$, we will sum together over all lengths quantities of the form $(\text{length}) \times (\# \text{ strings of that length})$. ...
Since $\sum_{\ell=1}^{k} 2^\ell = \sum_{\ell=0}^{k} 2^\ell - 1 = 2^{k+1} - 2$, thus sum has the form:

(29.10)
Since $\sum_{\ell=1}^{k} 2^{\ell} = \sum_{\ell=0}^{k} 2^{\ell} - 1 = 2^{k+1} - 2$, thus sum has the form:

\[ n_k \]

(29.10)
proof of Lemma 29.5.3 continued.

Since \( \sum_{\ell=1}^{k} 2^{\ell} = \sum_{\ell=0}^{k} 2^{\ell} - 1 = 2^{k+1} - 2 \), thus sum has the form:

\[
n_k = \sum_{j=1}^{k} j2^j
\]

(29.10)
proof of Lemma 29.5.3 continued.

Since \( \sum_{\ell=1}^{k} 2^\ell = \sum_{\ell=0}^{k} 2^\ell - 1 = 2^{k+1} - 2 \), thus sum has the form:

\[
\sum_{j=1}^{k} j 2^j = \sum_{j=1}^{k} 2^j + \sum_{j=2}^{k} 2^j + \cdots + \sum_{j=k}^{k} 2^j
\]

(29.10)

Now, consider a length \( n \) binary string. Number of distinct phrases \( c \) in that string is maximized when all phrases are as short as possible.
proof of Lemma 29.5.3 continued.

Since \( \sum_{\ell=1}^{k} 2^\ell = \sum_{\ell=0}^{k} 2^\ell - 1 = 2^{k+1} - 2 \), thus sum has the form:

\[
n_k = \sum_{j=1}^{k} j 2^j = \sum_{j=1}^{k} 2^j + \sum_{j=2}^{k} 2^j + \cdots + \sum_{j=k}^{k} 2^j = \sum_{\ell=1}^{k} \sum_{j=\ell}^{k} 2^j \tag{29.8}
\]

\[
(29.10)
\]

Now, consider a length \( n \) binary string. Number of distinct phrases \( c \) in that string is maximized when all phrases are as short as possible.
... proof of Lemma 29.5.3 continued.

Since \( \sum_{\ell=1}^{k} 2^\ell = \sum_{\ell=0}^{k} 2^\ell - 1 = 2^{k+1} - 2 \), thus sum has the form:

\[
n_k = \sum_{j=1}^{k} j 2^j = \sum_{j=1}^{k} 2^j + \sum_{j=2}^{k} 2^j + \cdots + \sum_{j=k}^{k} 2^j = \sum_{\ell=1}^{k} \sum_{j=\ell}^{k} 2^j \quad (29.8)
\]

\[
= \sum_{\ell=1}^{k} \left( \sum_{j=1}^{k} 2^j - \sum_{k=1}^{\ell-1} 2^k \right) \quad (29.10)
\]
Since $\sum_{\ell=1}^{k} 2^\ell = \sum_{\ell=0}^{k} 2^\ell - 1 = 2^{k+1} - 2$, thus sum has the form:

$$n_k = \sum_{j=1}^{k} j 2^j = \sum_{j=1}^{k} 2^j + \sum_{j=2}^{k} 2^j + \cdots + \sum_{j=k}^{k} 2^j = \sum_{\ell=1}^{k} \sum_{j=\ell}^{k} 2^j$$  \hspace{1cm} (29.8)

$$= \sum_{\ell=1}^{k} \left( \sum_{j=1}^{k} 2^j - \sum_{k=1}^{\ell-1} 2^k \right) = \sum_{\ell=1}^{k} \left( 2^{k+1} - 2^\ell \right)$$  \hspace{1cm} (29.9)

$$= \sum_{\ell=1}^{k} (2^{k+1} - 2^\ell)$$  \hspace{1cm} (29.10)
Since $\sum_{\ell=1}^{k} 2^{\ell} = \sum_{\ell=0}^{k} 2^{\ell} - 1 = 2^{k+1} - 2$, thus sum has the form:

$$n_k = \sum_{j=1}^{k} j 2^j = \sum_{j=1}^{k} 2^j + \sum_{j=2}^{k} 2^j + \cdots + \sum_{j=k}^{k} 2^j = \sum_{\ell=1}^{k} \sum_{j=\ell}^{k} 2^j \quad (29.8)$$

$$= \sum_{\ell=1}^{k} \left( \sum_{j=1}^{k} 2^j - \sum_{k=1}^{\ell-1} 2^k \right) = \sum_{\ell=1}^{k} \left( 2^{k+1} - 2^\ell \right) \quad (29.9)$$

$$= k 2^{k+1} - (2^{k+1} - 2) \quad (29.10)$$
Since \( \sum_{\ell=1}^{k} 2^\ell = \sum_{\ell=0}^{k} 2^\ell - 1 = 2^{k+1} - 2 \), thus sum has the form:

\[
n_k = k \sum_{j=1}^{k} j 2^j = \sum_{j=1}^{k} 2^j + \sum_{j=2}^{k} 2^j + \cdots + \sum_{j=k}^{k} 2^j = \sum_{\ell=1}^{k} \sum_{j=\ell}^{k} 2^j \tag{29.8}
\]

\[
= \sum_{\ell=1}^{k} \left( \sum_{j=1}^{k} 2^j - \sum_{k=1}^{\ell-1} 2^k \right) = \sum_{\ell=1}^{k} \left( 2^{k+1} - 2^\ell \right) \tag{29.9}
\]

\[
= k2^{k+1} - (2^{k+1} - 2) = (k - 1)2^{k+1} + 2 \tag{29.10}
\]
\[ \sum_{\ell=1}^{k} 2^\ell = \sum_{\ell=0}^{k} 2^\ell - 1 = 2^{k+1} - 2, \]

Thus sum has the form:

\[ n_k = \sum_{j=1}^{k} j 2^j = \sum_{j=1}^{k} 2^j + \sum_{j=2}^{k} 2^j + \cdots + \sum_{j=k}^{k} 2^j = \sum_{\ell=1}^{k} \sum_{j=\ell}^{k} 2^j \quad (29.8) \]

\[ = \sum_{\ell=1}^{k} \left( \sum_{j=1}^{k} 2^j - \sum_{k=1}^{\ell-1} 2^k \right) = \sum_{\ell=1}^{k} \left( 2^{k+1} - 2^\ell \right) \quad (29.9) \]

\[ = k 2^{k+1} - (2^{k+1} - 2) = (k - 1)2^{k+1} + 2 = \eta_k \quad (29.10) \]

Now, consider a length \( n \) binary string.

\[ \eta = \eta_k \] for some \( k \)

\[ \eta_k \ll \eta_{k+1} \] for some \( k \)
proof of Lemma 29.5.3 continued.

- Since \( \sum_{\ell=1}^{k} 2^\ell = \sum_{\ell=0}^{k} 2^\ell - 1 = 2^{k+1} - 2 \), thus sum has the form:

\[
 n_k = \sum_{j=1}^{k} j2^j = \sum_{j=1}^{k} 2^j + \sum_{j=2}^{k} 2^j + \cdots + \sum_{j=k}^{k} 2^j = \sum_{\ell=1}^{k} \sum_{j=1}^{\ell} 2^j \tag{29.8}
\]

\[
 = \sum_{\ell=1}^{k} \left( \sum_{j=1}^{k} 2^j - \sum_{k=1}^{\ell-1} 2^k \right) = \sum_{\ell=1}^{k} \left( 2^{k+1} - 2^\ell \right) \tag{29.9}
\]

\[
 = k2^{k+1} - (2^{k+1} - 2) = (k - 1)2^{k+1} + 2 \tag{29.10}
\]

- Now, consider a length \( n \) binary string.

- Number of distinct phrases \( c \) in that string is maximized when all phrases are as short as possible.
If $n = n_k$ (i.e., we consider a string of length $n$ equal to the sum of lengths of all distinct strings of length \( \leq k \)), then $c$ is maximized by considering all distinct strings of length \( \leq k \).
If \( n = n_k \) (i.e., we consider a string of length \( n \) equal to the sum of lengths of all distinct strings of length \( \leq k \)), then \( c \) is maximized by considering all distinct strings of length \( \leq k \).

I.e., when \( n = n_k \) (and recall \( n_k = (k-1)2^{k+1} + 2 \)), then number of distinct strings of length \( \leq k \) is

\[
c(n) = c(n_k) \leq \sum_{j=1}^{k} 2^j = 2^{k+1} - 2 \leq 2^{k+1} \leq \frac{n_k}{k-1} = \frac{n}{k-1}
\]

(29.11)
proof of Lemma 29.5.3 continued.

- If \( n = n_k \) (i.e., we consider a string of length \( n \) equal to the sum of lengths of all distinct strings of length \( \leq k \)), then \( c \) is maximized by considering all distinct strings of length \( \leq k \).

- I.e., when \( n = n_k \) (and recall \( n_k = (k - 1)2^{k+1} + 2 \)), then number of distinct strings of length \( \leq k \) is

\[
c(n) = c(n_k) \leq \sum_{j=1}^{k} 2^j = 2^{k+1} - 2 \leq 2^{k+1} \leq \frac{n_k}{k - 1} = \frac{n}{k - 1}
\]

(29.11)

- Note, \( n_k + (k + 1)2^{k+1} = 2k2^{k+1} + 2 = k2^{k+2} + 2 = n_{k+1} \)
...proof of Lemma 29.5.3 continued.

- If $n = n_k$ (i.e., we consider a string of length $n$ equal to the sum of lengths of all distinct strings of length $\leq k$), then $c$ is maximized by considering all distinct strings of length $\leq k$.

- I.e., when $n = n_k$ (and recall $n_k = (k - 1)2^{k+1} + 2$), then number of distinct strings of length $\leq k$ is

$$c(n) = c(n_k) \leq \sum_{j=1}^{k} 2^j = 2^{k+1} - 2 \leq 2^{k+1} \leq \frac{n_k}{k-1} = \frac{n}{k-1}$$

(29.11)

- Note, $n_k + (k + 1)2^{k+1} = 2k2^{k+1} + 2 = k2^{k+2} + 2 = n_{k+1}$

- If, on the other hand, suppose $n_k \leq n < n_{k+1}$, say $n = n_k + \delta$ with $\delta < (k + 1)2^{k+1} = n_{k+1} - n_k$ (which is the next step of $n_k$). ...
... proof of Lemma 29.5.3 continued.

- In such case, if we parse a length $n_k \leq n < n_{k+1}$ string into shortest possible distinct phrases, then $c(n)$ is maximized by:

$$c(n) \leq n_k k - 1 + \frac{\delta}{k+1} = n_k k - \delta k - 1$$

(29.12)
... proof of Lemma 29.5.3 continued.

In such case, if we parse a length $n_k \leq n < n_{k+1}$ string into shortest possible distinct phrases, then $c(n)$ is maximized by:

1. certainly not using phrases longer than $k + 1$, 

...
... proof of Lemma 29.5.3 continued.

In such case, if we parse a length $n_k \leq n < n_{k+1}$ string into shortest possible distinct phrases, then $c(n)$ is maximized by:

1. certainly not using phrases longer than $k + 1$,
2. and using up each and every phrase of length $\leq k$ and there are $c(n_k) \leq n/k + 1$ of them,
Lempel-Ziv Proof

... proof of Lemma 29.5.3 continued.

In such case, if we parse a length $n_k \leq n < n_{k+1}$ string into shortest possible distinct phrases, then $c(n)$ is maximized by:

1. certainly not using phrases longer than $k + 1$,
2. and using up each and every phrase of length $\leq k$ and there are $c(n_k) \leq n/k + 1$ of them,
3. and the remaining $n - n_k = \delta$ of the string is parsed into unique phrases of length $k + 1$, so there are $\delta/(k + 1)$ of them.
In such case, if we parse a length $n_k \leq n < n_{k+1}$ string into shortest possible distinct phrases, then $c(n)$ is maximized by:

1. certainly not using phrases longer than $k + 1$,
2. and using up each and every phrase of length $\leq k$ and there are $c(n_k) \leq n/k + 1$ of them,
3. and the remaining $n - n_k = \delta$ of the string is parsed into unique phrases of length $k + 1$, so there are $\delta/(k + 1)$ of them.

Then,

$$c(n) \leq \frac{n_k}{k-1} + \frac{\delta}{k+1} \leq \frac{n_k + \delta}{k-1} = \frac{n}{k-1}$$

(29.12)

where $k$ is such that $n_k \leq n < n_{k+1}$.
...proof of Lemma 29.5.3 continued.

- So for a given $n$, let's assume $k$ is such that $n_k \leq n < n_{k+1}$.
... proof of Lemma 29.5.3 continued.

- So for a given $n$, let's assume $k$ is such that $n_k \leq n < n_{k+1}$.
- Next, we bound $k$ given an $n$ (recall $n$ is string length, $k$ is max phrase length).
So for a given $n$, let's assume $k$ is such that $n_k \leq n < n_{k+1}$.

Next, we bound $k$ given an $n$ (recall $n$ is string length, $k$ is max phrase length).

Since $n \geq n_k = (k - 1)2^{k+1} + 2 \geq 2^k$, we have $k \leq \log n$. 

... proof of Lemma 29.5.3 continued.
... proof of Lemma 29.5.3 continued.

- So for a given $n$, let assume $k$ is such that $n_k \leq n < n_{k+1}$.
- Next, we bound $k$ given an $n$ (recall $n$ is string length, $k$ is max phrase length).
- Since $n \geq n_k = (k - 1)2^{k+1} + 2 \geq 2^k$, we have $k \leq \log n$
- Also,

$$n < n_{k+1} = k2^{k+2} + 2 \leq (k + 2)2^{k+2} \leq (\log n + 2)2^{k+2} \quad (29.13)$$

...
proof of Lemma 29.5.3 continued.

- So for a given \( n \), let's assume \( k \) is such that \( n_k \leq n < n_{k+1} \).
- Next, we bound \( k \) given an \( n \) (recall \( n \) is string length, \( k \) is max phrase length).
- Since \( n \geq n_k = (k - 1)2^{k+1} + 2 \geq 2^k \), we have \( k \leq \log n \)
- Also,

\[
    n < n_{k+1} = k2^{k+2} + 2 \leq (k + 2)2^{k+2} \leq (\log n + 2)2^{k+2} \tag{29.13}
\]

- which implies

\[
    k + 2 \geq \log \left( \frac{n}{\log n + 2} \right) \tag{29.14}
\]

...
Lempel-Ziv Proof

... proof of Lemma 29.5.3 continued.

- Now, for $n \geq 4$, we have

\[
\log n \geq 2 \quad (29.17)
\]

where

\[
\epsilon_n = \min \left(1, \log \log n + 4 \frac{\log n}{\log n} \right) \to 0 \quad (29.20)
\]
Now, for $n \geq 4$, we have

$$k - 1$$

(29.19)
Now, for $n \geq 4$, we have

$$k - 1 \geq \log n - \log(\log n + 2) - 3$$

by prev. inequality

(29.15)

where

$$\epsilon_n = \min \left( 1, \log \log n + \frac{4}{\log n} \right) \rightarrow 0$$

(29.19)
Lempel-Ziv Proof

... proof of Lemma 29.5.3 continued.

Now, for \( n \geq 4 \), we have

\[
k - 1 \geq \log n - \log(\log n + 2) - 3
\]

by prev. inequality

\[
= \left(1 - \frac{\log(\log n + 2) + 3}{\log n}\right) \log n
\]

take out factor \( \log n \)

(29.15)

(29.16)

(29.19)
Now, for $n \geq 4$, we have

$$k - 1 \geq \log n - \log(\log n + 2) - 3$$

by prev. inequality

$$= \left(1 - \frac{\log(\log n + 2) + 3}{\log n}\right) \log n$$

$$\geq \left(1 - \frac{\log(2 \log n) + 3}{\log n}\right) \log n$$

take out factor $\log n$

$$n \geq 4 \Rightarrow \log n \geq 2$$

$$\Rightarrow \log n \geq 2$$

(29.15)

(29.16)

(29.17)

(29.19)
Now, for $n \geq 4$, we have

$$k - 1 \geq \log n - \log(\log n + 2) - 3$$

by prev. inequality

$$= \left(1 - \frac{\log(\log n + 2) + 3}{\log n}\right) \log n$$

$$\geq \left(1 - \frac{\log(2 \log n) + 3}{\log n}\right) \log n$$

$$\geq \left(1 - \frac{\log(\log n) + 4}{\log n}\right) \log n$$

$n \geq 4 \Rightarrow \log n \geq 2$
Lempel-Ziv Proof

...proof of Lemma 29.5.3 continued.

Now, for $n \geq 4$, we have

$$k - 1 \geq \log n - \log(\log n + 2) - 3$$

by prev. inequality

$$= \left(1 - \frac{\log(\log n + 2) + 3}{\log n}\right) \log n$$

(29.15)

$$\geq \left(1 - \frac{\log(2 \log n) + 3}{\log n}\right) \log n$$

(29.16)

$$n \geq 4 \Rightarrow \log n \geq 2$$

(29.17)

$$= \left(1 - \frac{\log(\log n) + 4}{\log n}\right) \log n$$

(29.18)

$$= (1 - \epsilon_n) \log n$$

(29.19)
Now, for $n \geq 4$, we have

$$k - 1 \geq \log n - \log(\log n + 2) - 3$$

by prev. inequality (29.15)

$$= \left(1 - \frac{\log(\log n + 2) + 3}{\log n}\right) \log n$$

take out factor $\log n$ (29.16)

$$\geq \left(1 - \frac{\log(2 \log n) + 3}{\log n}\right) \log n$$

$n \geq 4 \Rightarrow \log n \geq 2$ (29.17)

$$= \left(1 - \frac{\log(\log n) + 4}{\log n}\right) \log n$$

(29.18)

$$= (1 - \epsilon_n) \log n$$

(29.19)

where

$$\epsilon_n = \min(1, \frac{\log \log n + 4}{\log n}) \rightarrow 0$$

(29.20)
Using this, and the fact that $k - 1 \geq 0$, and also from earlier that

$$c(n) \leq \frac{n}{k - 1}$$

we have that

$$c(n) \leq \frac{n}{k - 1} \leq \frac{n}{(1 - \epsilon_n) \log n} = \frac{n}{\log n} (1 + o(1))$$
Maximum Entropy Distribution (from Chapter 12)

Theorem 29.5.4

Let \( f \) be a probability density function with support set \( S \) (i.e., \( \int_{S} f(x) \, dx = 1 \)) satisfying the following \( m \) moment-matching constraints.

\[
\int_{S} f(x) r_i(x) = \alpha_i \quad \text{for} \quad 1 \leq i \leq m \quad (29.23)
\]

Then taking a density of the form \( f_\lambda(x) = e^{\lambda_0 + \sum_{i=1}^{m} \lambda_i r_i(x)} \) where \( \lambda = (\lambda_0, \ldots, \lambda_m) \) is chosen so that \( f_\lambda(x) \) satisfies the constraints, is the unique distribution satisfying the constraints that maximizes the differential entropy \( h(f) \).
**Maximum Entropy Distribution (from Chapter 12)**

**Theorem 29.5.4**

Let $f$ be a probability density function with support set $S$ (i.e., $\int_S f(x)dx = 1$) satisfying the following $m$ moment-matching constraints.

$$\int_S f(x) r_i(x) = \alpha_i \quad \text{for } 1 \leq i \leq m \quad (29.23)$$

Then taking a density of the form $f_\lambda(x) = e^{\lambda_0 + \sum_{i=1}^m \lambda_i r_i(x)}$ where $\lambda = (\lambda_0, \ldots, \lambda_m)$ is chosen so that $f_\lambda(x)$ satisfies the constraints, is the unique distribution satisfying the constraints that maximizes the differential entropy $h(f)$.

- This means that once we have a density in exponential form as $f_\lambda(x)$, then satisfying the constraints is sufficient to produce the maximum entropy distribution. Proof using Lagrange multipliers.
Theorem 29.5.4

Let $f$ be a probability density function with support set $S$ (i.e., $\int_S f(x) dx = 1$) satisfying the following $m$ moment-matching constraints.

$$\int_S f(x) r_i(x) = \alpha_i \quad \text{for } 1 \leq i \leq m \quad (29.23)$$

Then taking a density of the form $f_\lambda(x) = e^{\lambda_0 + \sum_{i=1}^{m} \lambda_i r_i(x)}$ where $\lambda = (\lambda_0, \ldots, \lambda_m)$ is chosen so that $f_\lambda(x)$ satisfies the constraints, is the unique distribution satisfying the constraints that maximizes the differential entropy $h(f)$.

- This means that once we have a density in exponential form as $f_\lambda(x)$, then satisfying the constraints is sufficient to produce the maximum entropy distribution. Proof using Lagrange multipliers.
- Ex: Gaussian distribution with a given mean and covariance is the unique maximum entropy distribution satisfying those first and second moment conditions.
Lemma 29.5.5

Let $Z$ be a positive integer valued random variable with mean $\mu$. Then we can bound $H(Z)$ as:

$$H(Z) \leq (\mu + 1) \log(\mu + 1) - \mu \log \mu$$  \hfill (29.24)
Lemma 29.5.5

Let $Z$ be a positive integer valued random variable with mean $\mu$. Then we can bound $H(Z)$ as:

$$H(Z) \leq (\mu + 1) \log(\mu + 1) - \mu \log \mu$$  (29.24)

Proof.

- A geometric distributed random variable $Z$ (defined on integers $1, 2, \ldots$) with mean $\mu = 1/p$ is defined as $P(Z = z) = (1 - p)^k p$. 

---

Prof. Jeff Bilmes
Simple Lemma

**Lemma 29.5.5**

Let $Z$ be a positive integer valued random variable with mean $\mu$. Then we can bound $H(Z)$ as:

$$H(Z) \leq (\mu + 1) \log(\mu + 1) - \mu \log \mu$$  \hspace{1cm} (29.24)

**Proof.**

- A geometric distributed random variable $Z$ (defined on integers $1, 2, \ldots$) with mean $\mu = 1/p$ is defined as $P(Z = z) = (1 - p)^k p$.
- The entropy of a geometric with mean $\mu$ is $(\mu + 1) \log(\mu + 1) - \mu \log \mu$
Lemma 29.5.5

Let $Z$ be a positive integer valued random variable with mean $\mu$. Then we can bound $H(Z)$ as:

$$H(Z) \leq (\mu + 1) \log(\mu + 1) - \mu \log \mu \quad (29.24)$$

Proof.

- A geometric distributed random variable $Z$ (defined on integers $1, 2, \ldots$) with mean $\mu = 1/p$ is defined as $P(Z = z) = (1 - p)^k p$.
- The entropy of a geometric with mean $\mu$ is $(\mu + 1) \log(\mu + 1) - \mu \log \mu$
- The geometric distribution is the distribution with maximum entropy over all positive-integer random variables with mean $\mu$ (again, using Lagrange multipliers)
Ergodicity Intuition and Definition

Intuition: Ergodic processes cannot be separated into different persistent modes of behavior.
**Ergodicity Intuition and Definition**

- **Intuition**: Ergodic processes cannot be separated into different persistent modes of behavior.
- **This yields the result that time average are the same as ensemble averages**
Ergodicity Intuition and Definition

- Intuition: Ergodic processes cannot be separated into different persistent modes of behavior.
- This yields the result that time average are the same as ensemble averages
- Let \( x = \{x_i\} \) be a sequence of source letters.
Ergodicity Intuition and Definition

- Intuition: Ergodic processes cannot be separated into different persistent modes of behavior.
- This yields the result that time average are the same as ensemble averages.
- Let \( x = \{x_i\} \) be a sequence of source letters.
- Let \( T^\ell x \) be the shifted sequence in time by \( \ell \) positions. I.e., if \( T^\ell x = x' \) then \( x'_i = x_{i+\ell}, \forall i \).
Ergodicity Intuition and Definition

- **Intuition**: Ergodic processes cannot be separated into different persistent modes of behavior.
- This yields the result that time averages are the same as ensemble averages.
- Let \( x = \{x_i\} \) be a sequence of source letters.
- Let \( T^\ell x \) be the shifted sequence in time by \( \ell \) positions. I.e., if \( T^\ell x = x' \) then \( x'_i = x_{i+\ell}, \forall i \).
- Let \( S \) be an infinite set of sequences of source letters, i.e., \( S = \{x: x \text{ is a sequence of source letters}\} \).
Ergodicity Intuition and Definition

- **Intuition**: Ergodic processes cannot be separated into different persistent modes of behavior.
- This yields the result that time average are the same as ensemble averages.
- Let \( x = \{ x_i \} \) be a sequence of source letters.
- Let \( T^\ell x \) be the shifted sequence in time by \( \ell \) positions. I.e., if \( T^\ell x = x' \) then \( x'_i = x_{i+\ell}, \forall i \).
- Let \( S \) be an infinite set of sequences of source letters, i.e., \( S = \{ x : x \text{ is a sequence of source letters} \} \).
- And \( T^\ell S \) is the set of all sequences shifted by \( \ell \) positions. I.e., if \( x' = T^\ell x \), then \( x' \in T^\ell S \) if \( x \in S \).
Ergodicity Intuition and Definition

- Intuition: Ergodic processes cannot be separated into different persistent modes of behavior.
- This yields the result that time average are the same as ensemble averages.
- Let $x = \{x_i\}$ be a sequence of source letters.
- Let $T^\ell x$ be the shifted sequence in time by $\ell$ positions. I.e., if $T^\ell x = x'$ then $x'_i = x_{i+\ell}, \forall i$.
- Let $S$ be an infinite set of sequences of source letters, i.e., $S = \{x : x$ is a sequence of source letters $\}$.
- And $T^\ell S$ is the set of all sequences shifted by $\ell$ positions. I.e., if $x' = T^\ell x$, then $x' \in T^\ell S$ if $x \in S$.
- **Definition: Invariant.** A set $S$ is invariant if $T^\ell S = S$ for all $\ell$. 
Ergodicity Intuition and Definition

- Intuition: Ergodic processes cannot be separated into different persistent modes of behavior.
- This yields the result that time average are the same as ensemble averages.
- Let $x = \{x_i\}$ be a sequence of source letters.
- Let $T^\ell x$ be the shifted sequence in time by $\ell$ positions. I.e., if $T^\ell x = x'$ then $x'_i = x_{i+\ell}$, $\forall i$.
- Let $S$ be an infinite set of sequences of source letters, i.e., $S = \{x : x$ is a sequence of source letters $\}$.
- And $T^\ell S$ is the set of all sequences shifted by $\ell$ positions. I.e., if $x' = T^\ell x$, then $x' \in T^\ell S$ if $x \in S$.
- Definition: Invariant. A set $S$ is invariant if $T^\ell S = S$ for all $\ell$.
- Example: The set of all sequences of a discrete alphabet source is invariant.
Ergodicity Intuition and Definition

- Intuition: Ergodic processes cannot be separated into different persistent modes of behavior.
- This yields the result that time average are the same as ensemble averages.
- Let $x = \{x_i\}$ be a sequence of source letters.
- Let $T^\ell x$ be the shifted sequence in time by $\ell$ positions. I.e., if $T^\ell x = x'$ then $x'_i = x_{i+\ell}, \forall i$.
- Let $S$ be an infinite set of sequences of source letters, i.e., $S = \{x : x$ is a sequence of source letters $\}$.
- And $T^\ell S$ is the set of all sequences shifted by $\ell$ positions. I.e., if $x' = T^\ell x$, then $x' \in T^\ell S$ if $x \in S$.
- Definition: **Invariant**. A set $S$ is invariant if $T^\ell S = S$ for all $\ell$.
- Example: The set of all sequences of a discrete alphabet source is invariant.
- Example: $S = \{\ldots 000\ldots, \ldots 111\ldots\}$ is invariant.
Ergodicity Intuition and Definition

- **Ex:** For any sequence, the set

\[
\{ \ldots, T^{-2}x, T^{-1}x, T^0x, T^1x, T^2x, \ldots \},
\]

(29.25)

(which is the set of all shifts of a sequence) is invariant.
Ergodicity Intuition and Definition

- **Ex:** For any sequence, the set
  \[ \{ \ldots, T^{-2}x, T^{-1}x, T^{0}x, T^{1}x, T^{2}x, \ldots \} , \]  
  (which is the set of all shifts of a sequence) is invariant.

- **Definition:** A discrete stationary source is **ergodic** if every invariant set of sequences either has probability 1 or probability 0. I.e.,
  \[
  \Pr\{ S : T^{\ell}S = S, \forall \ell \} = 1 \text{ or } 0 \quad \forall S
  \]  
  (29.26)
Ergodicity Intuition and Definition

- Ex: For any sequence, the set
  \[
  \{\ldots, T^{-2}x, T^{-1}x, T^0x, T^{1}x, T^{2}x, \ldots\},
  \]  
  (which is the set of all shifts of a sequence) is invariant.

- Definition: A discrete stationary source is ergodic if every invariant set of sequences either has probability 1 or probability 0. I.e.,
  \[
  \Pr\left\{ S : T^\ell S = S, \forall \ell \right\} = 1 \text{ or } 0 \text{ } \forall S
  \]  
  (29.26)

- This implies that time averages give us ensemble averages.
Let $\{X_i\}_{i=-\infty}^{\infty}$ be a stationary ergodic process with probability mass function $p(x_1, x_2, \ldots, x_n)$.
Let \( \{X_i\}_{i=-\infty}^{\infty} \) be a stationary ergodic process with probability mass function \( p(x_1, x_2, \ldots, x_n) \).

For fixed integer \( k \), define \( k \)'th order Markov approximation to \( p \) as:

\[
Q_k(x_{-(k-1)}, \ldots, x_0, x_1, \ldots, x_n) \triangleq p(x_{-(k-1):0}) \prod_{j=1}^{n} p(x_j | x_{j-k:j-1})
\]  

(29.27)

where we think of \( x_{-(k-1):0} \) as state “0” and \( x_{j-k:j-1} \) as state “\( j \)”, and where \( x_{i:j} = \{x_i, x_{i+1}, \ldots, x_j\} \) with \( i \leq j \).
Let \( \{X_i\}_{i=-\infty}^{\infty} \) be a stationary ergodic process with probability mass function \( p(x_1, x_2, \ldots, x_n) \).

For fixed integer \( k \), define \( k \)'th order Markov approximation to \( p \) as:

\[
Q_{k}(x_{-(k-1)}, \ldots, x_0, x_1, \ldots, x_n) = p(x_{-(k-1):0}) \prod_{j=1}^{n} p(x_j|x_{j-k:j-1})
\]

where we think of \( x_{-(k-1):0} \) as state “0” and \( x_{j-k:j-1} \) as state “\( j \)”, and where \( x_{i:j} = \{x_i, x_{i+1}, \ldots, x_j\} \) with \( i \leq j \).

Note also that \( p(x_n|x_{n-k:n-1}) \) is also stationary and ergodic since \( p(x_{n-k:n}) \) also is.
We then get:

\[ \frac{1}{n} \log Q_k(x_1, \ldots, x_n| x_1) = \sum_{j=1}^{n} \log p(x_j|x_j-k:j-1) \]  

(29.28)

\[ \rightarrow -E \log p(X_j|X_j-k:j-1) = H(X_j|X_j-k:j-1) \]  

(29.29)

where \[ H(X) \] is the entropy rate of the stochastic process.
We then get:

\[- \frac{1}{n} \log Q_k(x_1, \ldots, x_n|x-(k-1):0)\]

(29.28)

\[- \frac{1}{n} \sum_{j=1}^{n} \log p(x_j|x_j-k:j-1)\]

(29.29)

which follows since the process is stationary ergodic.

We will show that:

\[\limsup_{n \to \infty} c(n) \log c(n) / n \leq H(x_j|x_j-k:j-1) \to H(x)\]

(29.30)

where \(H(x)\) is the entropy rate of the stochastic process.
$k$th order Markov chain approximation

We then get:

$$- \frac{1}{n} \log Q_k(x_1, \ldots, x_n|x_{-(k-1):0}) = - \frac{1}{n} \sum_{j=1}^{n} \log p(x_j|x_{j-k:j-1})$$

(29.28)

(29.29)
We then get:

\[- \frac{1}{n} \log Q_k(x_1, \ldots, x_n|x_{-(k-1):0}) = - \frac{1}{n} \sum_{j=1}^{n} \log p(x_j|x_{j-k:j-1})\]

(29.28)

\[\rightarrow - E \log p(X_j|X_{j-k:j-1})\]

(29.29)
We then get:

$$- \frac{1}{n} \log Q_k(x_1, \ldots, x_n|x_{-(k-1):0}) = - \frac{1}{n} \sum_{j=1}^{n} \log p(x_j|x_{j-k:j-1})$$

(29.28)

$$\rightarrow -E \log p(X_j|X_{j-k:j-1}) = H(X_j|X_{j-k:j-1})$$

(29.29)
We then get:

\[-\frac{1}{n} \log Q_k(x_1, \ldots, x_n|x_{-(k-1):0}) = -\frac{1}{n} \sum_{j=1}^{n} \log p(x_j|x_{j-k:j-1})\]

(29.28)

\[-E \log p(X_j|X_{j-k:j-1}) = H(X_j|X_{j-k:j-1})\]

(29.29)

which follows since the process is stationary ergodic.
We then get:

\[-\frac{1}{n} \log Q_k(x_1, \ldots, x_n|x_{-(k-1):0}) = -\frac{1}{n} \sum_{j=1}^{n} \log p(x_j|x_{j-k:j-1})\]

\[\rightarrow -E \log p(X_j|X_{j-k:j-1}) = H(X_j|X_{j-k:j-1})\] (29.28)

which follows since the process is stationary ergodic.

We will show that:

\[\limsup_{n \to \infty} \frac{c(n) \log c(n)}{n} \leq H(X_j|X_{j-k:j-1}) \to H(X)\] (29.30)

where \(H(X)\) is the entropy rate of the stochastic process.
$k$th order Markov chain approximation

- We then get:

$$\frac{1}{n} \log Q_k(x_1, \ldots, x_n|x_{-(k-1):0}) = -\frac{1}{n} \sum_{j=1}^{n} \log p(x_j|x_{j-k:j-1})$$

(29.28)

$$\rightarrow -E \log p(X_j|X_{j-k:j-1}) = H(X_j|X_{j-k:j-1})$$

(29.29)

which follows since the process is stationary ergodic.

- We will show that:

$$\limsup_{n \rightarrow \infty} \frac{c(n) \log c(n)}{n} \leq H(X_j|X_{j-k:j-1}) \rightarrow H(\mathcal{X})$$

(29.30)

where $H(\mathcal{X})$ is the entropy rate of the stochastic process. But first

...
...a bit more notation

- We consider $x_{1:n}$ parsed into $c$ distinct phrases, $x_i \in \mathcal{X}$, as follows

$$x_{1:n} = y_1 y_2 \ldots y_c$$

where $y_c$ is a subsequence.
...a bit more notation

- We consider $x_{1:n}$ parsed into $c$ distinct phrases, $x_i \in \mathcal{X}$, as follows $x_{1:n} = y_1y_2 \ldots y_c$ where $y_c$ is a subsequence.
- Let $v_i$ be the index into $x_{1:n}$ of the start of the $i$'th phrase. I.e., $y_i = x_{v_i:v_i+1-1}$ for all $i = 1, 2, \ldots, c$. 

Let $c_\ell^s$ be the number of phrases in $x_{1:n}$ with length $\ell$ and that have preceding state $s$, for $\ell = 1, 2, \ldots, n$ and $s \in \mathcal{X}^k$ (length $k$ strings).

I.e., $c_\ell^s = |\{i : i \in \{1, \ldots, c\}, |y_i| = \ell, s_i = s\}|$ (29.31)
We consider $x_{1:n}$ parsed into $c$ distinct phrases, $x_i \in \mathcal{X}$, as follows

$$x_{1:n} = y_1 y_2 \ldots y_c$$

where $y_c$ is a subsequence.

Let $v_i$ be the index into $x_{1:n}$ of the start of the $i$'th phrase. I.e.,

$$y_i = x_{v_i:v_{i+1}-1} \text{ for all } i = 1, 2, \ldots, c.$$ 

Let $s_i = x_{v_i-k:v_i-1}$ which is the $k$ symbols of $x_{1:n}$ preceding $y_i$ with $s_1 = x_{-(k-1):0}$, so $s_i$ is the “state” or the prefix of the $i$’th phrase.
We consider \( x_{1:n} \) parsed into \( c \) distinct phrases, \( x_i \in \mathcal{X} \), as follows
\[
x_{1:n} = y_1 y_2 \ldots y_c \text{ where } y_c \text{ is a subsequence.}
\]

Let \( v_i \) be the index into \( x_{1:n} \) of the start of the \( i \)'th phrase. I.e.,
\[
y_i = x_{v_i:v_i+1-1} \text{ for all } i = 1, 2, \ldots, c.
\]

Let \( s_i = x_{v_i-k:v_i-1} \) which is the \( k \) symbols of \( x_{1:n} \) preceding \( y_i \) with
\[
s_1 = x_{-(k-1):0}, \text{ so } s_i \text{ is the “state” or the prefix of the } i \text{’th phrase.}
\]

Let \( c_{\ell s} \) be the number of phrases in \( x_{1:n} \) with length \( \ell \) and that have preceding state \( s \), for \( \ell = 1, 2, \ldots, n \) and \( s \in \mathcal{X}^k \) (length \( k \) strings). I.e., \( c_{\ell s} \) is the number of phrases of length \( \ell \) with prefix \( s \),
... a bit more notation

- We consider \( x_{1:n} \) parsed into \( c \) distinct phrases, \( x_i \in \mathcal{X} \), as follows
  \[ x_{1:n} = y_1y_2 \ldots y_c \] where \( y_c \) is a subsequence.

- Let \( v_i \) be the index into \( x_{1:n} \) of the start of the \( i \)'th phrase. I.e.,
  \[ y_i = x_{v_i:v_i+1-1} \] for all \( i = 1, 2, \ldots, c \).

- Let \( s_i = x_{v_i-k:v_i-1} \) which is the \( k \) symbols of \( x_{1:n} \) preceding \( y_i \) with
  \[ s_1 = x_{-(k-1):0} \], so \( s_i \) is the “state” or the prefix of the \( i \)'th phrase.

- Let \( c_{\ell,s} \) be the number of phrases in \( x_{1:n} \) with length \( \ell \) and that have
  preceding state \( s \), for \( \ell = 1, 2, \ldots, n \) and \( s \in \mathcal{X}^k \) (length \( k \) strings).
  I.e., \( c_{\ell,s} \) is the number of phrases of length \( \ell \) with prefix \( s \), or,

  \[ c_{\ell,s} = |\{i : i \in \{1, \ldots, c\}, |y_i| = \ell, s_i = s\}| \quad (29.31) \]
... a bit more notation

- We consider $x_{1:n}$ parsed into $c$ distinct phrases, $x_i \in \mathcal{X}$, as follows
  $x_{1:n} = y_1 y_2 \ldots y_c$ where $y_c$ is a subsequence.

- Let $v_i$ be the index into $x_{1:n}$ of the start of the $i$’th phrase. I.e.,
  $y_i = x_{v_i:v_i+1-1}$ for all $i = 1, 2, \ldots, c$.

- Let $s_i = x_{v_i-k:v_i-1}$ which is the $k$ symbols of $x_{1:n}$ preceding $y_i$ with
  $s_1 = x_{-(k-1):0}$, so $s_i$ is the “state” or the prefix of the $i$’th phrase.

- Let $c_{\ell s}$ be the number of phrases in $x_{1:n}$ with length $\ell$ and that have
  preceding state $s$, for $\ell = 1, 2, \ldots, n$ and $s \in \mathcal{X}^k$ (length $k$ strings).
  I.e., $c_{\ell s}$ is the number of phrases of length $\ell$ with prefix $s$, or,

  \[ c_{\ell s} = |\{i : i \in \{1, \ldots, c\}, |y_i| = \ell, s_i = s\}| \quad (29.31) \]

- Then $\sum_{\ell,s} c_{\ell s} = c = c(n)$, where $c = c(n)$ is the total number of
  phrases in a distinct parsing of a sequence of length $n$. 
...a bit more notation and a lemma

- And $\sum_{\ell,s} \ell c_{\ell s} = n$ which is the total string length.
And $\sum_{\ell,s} \ell c_{\ell s} = n$ which is the total string length.

Key question: Can we relate (or at least bound) the probability to some deterministic aspect of the parsing?
... a bit more notation and a lemma

- And $\sum_{\ell,s} \ell c_{\ell s} = n$ which is the total string length.
- Key question: Can we relate (or at least bound) the probability to some deterministic aspect of the parsing? Yes.
... a bit more notation and a lemma

- And $\sum_{\ell,s} \ell c_{\ell s} = n$ which is the total string length.
- Key question: Can we relate (or at least bound) the probability to some deterministic aspect of the parsing? Yes.
- We have the following lemma:
\[ ∑_{ℓ,s} ℓc_{ℓs} = n \] which is the total string length.

Key question: Can we relate (or at least bound) the probability to some deterministic aspect of the parsing? Yes.

We have the following lemma:

**Lemma 29.5.6 (Ziv’s inequality)**

*For any distinct parsing (which includes the LZ parsing) of the string \( x_{1:n} \), we have:*

\[
\log Q_k(x_1, x_2, \ldots x_n | s_1) \leq - ∑_{ℓ,s} c_{ℓs} \log c_{ℓs} \tag{29.32}
\]
And $\sum_{\ell,s} \ell c_{\ell s} = n$ which is the total string length.

Key question: Can we relate (or at least bound) the probability to some deterministic aspect of the parsing? Yes.

We have the following lemma:

**Lemma 29.5.6 (Ziv’s inequality)**

For any distinct parsing (which includes the LZ parsing) of the string $x_1:n$, we have:

$$\log Q_k(x_1, x_2, \ldots x_n | s_1) \leq -\sum_{\ell, s} c_{\ell s} \log c_{\ell s} \tag{29.32}$$

Note that the bound is independent of $Q$ and depends only on $c_{\ell s}$, which is the number of phrases of length $\ell$ with prefix (state) $s$. 

...a bit more notation and a lemma
...a bit more notation and a lemma

- And $\sum_{\ell,s} c_{\ell s} = n$ which is the total string length.
- Key question: Can we relate (or at least bound) the probability to some deterministic aspect of the parsing? Yes.
- We have the following lemma:

**Lemma 29.5.6 (Ziv’s inequality)**

*For any distinct parsing (which includes the LZ parsing) of the string $x_{1:n}$, we have:*

$$\log Q_k(x_1, x_2, \ldots x_n | s_1) \leq - \sum_{\ell,s} c_{\ell s} \log c_{\ell s}$$  \hspace{1cm} (29.32)

- Note that the bound is independent of $Q$ and depends only on $c_{\ell s}$, which is the number of phrases of length $\ell$ with prefix (state) $s$.
- Key idea: as there is more diversity in string $x_{1:n}$, the max possible probability decreases. “distinct” $y_i$’s increase diversity.
Proof of Ziv’s inequality

First, we have:

\[ Q_k(x_{1:n}|s_1) = Q_k(y_{1:c}|s_1) = \prod_{i=1}^{c} p(y_i|s_i) \]  

(29.33)

where the r.h.s. follows because of the \( k' \)th order Markov assumption, that \( y_i \) depends on nothing else in the past given the immediate past \( s_i \).
Proof of Ziv’s inequality

First, we have:

\[ Q_k(x_1:n|s_1) = Q_k(y_1:c|s_1) = \prod_{i=1}^{c} p(y_i|s_i) \]  \hspace{1cm} (29.33)

where the r.h.s. follows because of the \( k' \)th order Markov assumption, that \( y_i \) depends on nothing else in the past given the immediate past \( s_i \).

This gives

\[ \log Q_k(x_1, x_2, \ldots, x_n|s_1) = \sum_i \log p(y_i|s_i) \]  \hspace{1cm} (29.34)

\[ = \sum_{\ell,s} \sum_{i:|y_i|=\ell,s_i=s} \log p(y_i|s_i) = \sum_{\ell,s} c_{\ell s} \sum_{i:|y_i|=\ell,s_i=s} \frac{1}{c_{\ell s}} \log p(y_i|s_i) \]
Proof of Ziv’s inequality

- But we have mixture, since

\[ \sum_{i:|y_i| = \ell, s_i = s} \frac{1}{c_{\ell s}} = 1 \]  

(29.35)

since \( c_{\ell s} \) is the number of length \( \ell \) phrases with prefix \( s \), and the sum sums over all of them.

...
Proof of Ziv’s inequality

proof of Ziv’s inequality.

But we have mixture, since

$$\sum_{i: |y_i| = \ell, s_i = s} \frac{1}{c_{\ell s}} = 1 \quad (29.35)$$

since $c_{\ell s}$ is the number of length $\ell$ phrases with prefix $s$, and the sum sums over all of them.

So, using Jensen’s inequality, we get that:

$$\sum_{\ell, s} c_{\ell s} \sum_{i: |y_i| = \ell, s_i = s} \frac{1}{c_{\ell s}} \log p(y_i | s_i) \leq \sum_{\ell, s} c_{\ell s} \log \left( \sum_{i: |y_i| = \ell, s_i = s} \frac{1}{c_{\ell s}} p(y_i | s_i) \right) \quad (29.36)$$

...
Proof of Ziv’s inequality

But the $y_i$’s are distinct (no double counts) which means that

$$\sum_{i:y_i = \ell, s_i = s} p(y_i \mid s_i) \leq 1 \quad (29.37)$$
Proof of Ziv’s inequality

But the $y_i$’s are distinct (no double counts) which means that

$$\sum_{i: |y_i| = \ell, s_i = s} p(y_i | s_i) \leq 1 \quad (29.37)$$

this then yields our result, that

$$\log Q_k(x_1, x_2, \ldots, x_n | s_1) \leq \sum_{\ell, s} c_{\ell s} \log \left( \sum_{i: |y_i| = \ell, s_i = s} \frac{1}{c_{\ell s}} p(y_i | s_i) \right) \quad (29.38)$$

$$\leq \sum_{\ell, s} c_{\ell s} \log \left( \frac{1}{c_{\ell s}} \right) \quad (29.39)$$
Recall limsup/liminf (from lecture 28)

- Recall,

\[
\limsup_{n \to \infty} a_n \triangleq \inf_{n>0} \left( \sup_{k>n} a_k \right) = \inf S \tag{29.40}
\]

where

\[
S = \{ a : a = \sup B_n \text{ for some } n, \text{ with } B_n = \{ a_n, a_{n+1}, \ldots, \} \}.
\]
Recall \( \limsup \) and \( \liminf \) (from lecture 28)

- Recall, \( \limsup a_n \triangleq \inf_{n \to \infty} \left( \sup_{k > n} a_k \right) = \inf S \) (29.40)

where

\[ S = \{ a : a = \sup B_n \text{ for some } n, \text{ with } B_n = \{ a_n, a_{n+1}, \ldots, \} \} \]

- For example, while \( \lim_{x \to \infty} \sin(x) \) does not exist,
  \( \limsup_{x \to \infty} \sin(x) = 1 \).
Recall limsup and liminf (from lecture 28)

- Recall,

\[
\limsup_{n \to \infty} a_n \triangleq \inf_{n > 0} \left( \sup_{k > n} a_k \right) = \inf S
\]

where

\[ S = \{a : a = \sup B_n \text{ for some } n, \text{ with } B_n = \{a_n, a_{n+1}, \ldots, \} \}. \]

- For example, while \( \lim_{x \to \infty} \sin(x) \) does not exist,

\[ \limsup_{x \to \infty} \sin(x) = 1. \]

- Also, \( \limsup_{x \to \infty} (\sin(x) - \sin^2(x)) = \)
Recall limsup/liminf (from lecture 28)

- Recall,
  \[ \limsup_{n \to \infty} a_n \triangleq \inf_{n > 0} \left( \sup_{k > n} a_k \right) = \inf S \]  
  (29.40)

  where
  \[ S = \{ a : a = \sup B_n \text{ for some } n, \text{ with } B_n = \{ a_n, a_{n+1}, \ldots, \} \}. \]

- For example, while \( \lim_{x \to \infty} \sin(x) \) does not exist,
  \[ \limsup_{x \to \infty} \sin(x) = 1. \]

- Also, \( \limsup_{x \to \infty} (\sin(x) - \sin^2(x)) = \)
recall limsup/liminf (from lecture 28)

- Recall,

\[
\limsup_{n \to \infty} a_n \triangleq \inf_{n > 0} \left( \sup_{k > n} a_k \right) = \inf S
\]  

(29.40)

where

\[
S = \{ a : a = \sup B_n \text{ for some } n, \text{ with } B_n = \{a_n, a_{n+1}, \ldots, \} \}. 
\]

- For example, while \( \lim_{x \to \infty} \sin(x) \) does not exist,

\[
\limsup_{x \to \infty} \sin(x) = 1. 
\]

- Also, \( \limsup_{x \to \infty}(\sin(x) - \sin^2(x)) = 1/4. \)

- Thus, \( \limsup \) allows for oscillation in the sequences and in some sense \( \limsup \) asks for infimum convergence in the local maxima (or perhaps better, “reverse-time cumulative” local maxima).
Recall limsup/liminf (from lecture 28)

- Recall,

\[
\limsup_{n \to \infty} a_n \triangleq \inf_{n > 0} \left( \sup_{k > n} a_k \right) = \inf S
\]  

(29.40)

where

\[ S = \{ a : a = \sup B_n \text{ for some } n, \text{ with } B_n = \{ a_n, a_{n+1}, \ldots \} \}. \]

- For example, while \( \lim_{x \to \infty} \sin(x) \) does not exist, \( \limsup_{x \to \infty} \sin(x) = 1 \).

- Also, \( \limsup_{x \to \infty} (\sin(x) - \sin^2(x)) = 1/4 \).

- Thus, \( \limsup \) allows for oscillation in the sequences and in some sense \( \limsup \) asks for infimum convergence in the local maxima (or perhaps better, “reverse-time cumulative” local maxima).

- Also,

\[
\liminf_{n \to \infty} a_n \triangleq \sup_{n > 0} \left( \inf_{k > n} a_k \right)
\]  

(29.41)

so \( \liminf \) asks for supremum convergence in the local minima.
Main Theorem

**Theorem 29.5.7**

Let \( X_{1:n} \) be a stationary ergodic process with entropy rate \( H(\mathcal{X}) \), and \( c(n) \) be the number of phrases in a distinct parsing of a sample of length \( n \) from this process. Then

\[
\limsup_{n \to \infty} \frac{c(n) \log c(n)}{n} \leq H(\mathcal{X}) \quad \text{w.p.1} \tag{29.42}
\]

**Proof.**

- We write \( c \) for \( c(n) \).
Main Theorem

**Theorem 29.5.7**

Let $X_{1:n}$ be a stationary ergodic process with entropy rate $H(X)$, and $c(n)$ be the number of phrases in a distinct parsing of a sample of length $n$ from this process. Then

$$\limsup_{n \to \infty} \frac{c(n) \log c(n)}{n} \leq H(X) \quad \text{w.p.1} \quad (29.42)$$

**Proof.**

- We write $c$ for $c(n)$. By Ziv’s inequality, and since $\sum_{\ell,s} c_{\ell s} = c$:

$$\log Q_k(x_1, x_2, \ldots, x_n | s_1) \leq - \sum_{\ell,s} c_{\ell s} \log \frac{c_{\ell s} c}{c} \quad (29.43)$$

$$= -c \log c - c \sum_{\ell,s} \frac{c_{\ell s}}{c} \log \frac{c_{\ell s}}{c} \quad (29.44)$$

...
Let's write $\pi_{\ell s} = c_{\ell s}/c$, which can be treated as a probability since $\pi_{\ell s} \geq 0$ and $\sum_{\ell s} \pi_{\ell s} = 1$. Then since $\sum_{\ell s} c_{\ell s} = n$, we have $\sum_{\ell s} \ell \pi_{\ell s} = n/c$ (29.45)

Define new random variables $U, V$ s.t., $p(U = \ell, V = s) = \pi_{\ell s}$ (29.46)

So that $E(U) = \sum_{\ell} \ell \sum_{s} \pi_{\ell s} = n/c$ (29.47)
Main Theorem

... proof continued.

- Lets write $\pi_{\ell s} = c_{\ell s}/c$, which can be treated as a probability since $\pi_{\ell s} \geq 0$ and $\sum_{\ell s} \pi_{\ell s} = 1$.

- Then since $\sum_{\ell s} \ell c_{\ell s} = n$, we have

$$\sum_{\ell s} \ell \pi_{\ell s} = n/c$$  \(29.45\)
\[ \pi_{ls} = \frac{c_{ls}}{c}, \] which can be treated as a probability since \( \pi_{ls} \geq 0 \) and \( \sum_{ls} \pi_{ls} = 1 \).

Then since \( \sum_{ls} \ell c_{ls} = n \), we have

\[ \sum_{ls} \ell \pi_{ls} = \frac{n}{c} \quad (29.45) \]

Define new random variables \( U, V \) s.t.

\[ p(U = \ell, V = s) = \pi_{ls} \quad (29.46) \]
... proof continued.

- Let's write $\pi_{ls} = \frac{c_{ls}}{c}$, which can be treated as a probability since $\pi_{ls} \geq 0$ and $\sum_{ls} \pi_{ls} = 1$.
- Then since $\sum_{ls} \ell c_{ls} = n$, we have

$$\sum_{ls} \ell \pi_{ls} = n/c \quad (29.45)$$

- Define new random variables $U, V$ s.t.,

$$p(U = \ell, V = s) = \pi_{ls} \quad (29.46)$$

- So that

$$EU = \sum_{\ell} \ell \pi_{\ell} = \sum_{\ell} \ell \sum_{s} \pi_{ls} = n/c \quad (29.47)$$
Main Theorem

... proof continued.

This immediately gives us:

\[
\log Q_k(x_{1:n} | s_1) \leq \sum_{\ell_s} c_{\ell_s} \log 1/c_{\ell_s} \leq cH(U, V) - c \log c
\]  

(29.48)
Main Theorem

... proof continued.

- This immediately gives us:

\[
\log Q_k(x_1:n|s_1) \leq \sum_{\ell s} c_{\ell s} \log 1/c_{\ell s} \leq c H(U, V) - c \log c \tag{29.48}
\]

- Or

\[
-\frac{1}{n} \log Q_k(x_1:n|s_1) \geq \frac{c}{n} \log c - \frac{c}{n} H(U, V) \tag{29.49}
\]
Main Theorem

... proof continued.

- This immediately gives us:

\[
\log Q_k(x_1:n \mid s_1) \leq \sum_{\ell s} c_{\ell s} \log 1/c_{\ell s} \tag{29.48}
\]

\[
= cH(U, V) - c \log c \tag{29.49}
\]

- Or

\[
- \frac{1}{n} \log Q_k(x_1:n \mid s_1) \geq \frac{c}{n} \log c - \frac{c}{n} H(U, V)
\]

\[\rightarrow \text{entropy rate as } k \rightarrow \infty\]

and \( n \rightarrow \infty \)
Main Theorem

... proof continued.

- This immediately gives us:

\[
\log Q_k(x_1:n|s_1) \leq \sum_{\ell s} c_{\ell s} \log \frac{1}{c_{\ell s}} \quad (29.48)
\]

\[
= cH(U, V) - c \log c \quad (29.49)
\]

- Or

\[
-c = c(n), \text{ so this is what we wish to show converges to entropy of } X
\]

\[
\frac{1}{n} \log Q_k(x_1:n|s_1) \geq \frac{c}{n} \log c - \frac{c}{n} H(U, V)
\]

\[
\rightarrow \text{entropy rate as } k \to \infty
\]

and \( n \to \infty \)
Main Theorem

... proof continued.

- This immediately gives us:

\[
\log Q_k(x_1:n|s_1) \leq \sum_{\ell s} c_{\ell s} \log 1/c_{\ell s}
\]

\[
= c H(U, V) - c \log c
\] (29.48)

- Or

\[
- \frac{1}{n} \log Q_k(x_1:n|s_1) \geq \frac{c}{n} \log c - \frac{c}{n} H(U, V)
\]

\[
\rightarrow \text{entropy rate as } k \rightarrow \infty
\]

and \( n \rightarrow \infty \)

Ideally, this will \( \rightarrow 0 \) as \( n \rightarrow \infty \) (29.50)
Now, we know we have $H(U, V) \leq H(U) + H(V)$.

... proof continued.
Now, we know we have $H(U, V) \leq H(U) + H(V)$.

Also, we have $H(V) \leq \log |\{0, 1\}|^k = k$, we can think of $V$ as a state (binary string of length $k$) variable.
\[ H(U) \leq (EU + 1) \log(2EU) - EU \log EU \] (29.51)

\[ = (nc + 1) \log(nc + 1) - nc \log nc \] (29.52)

\[ = nc \log(nc + 1) + \log(nc + 1) \] (29.53)

\[ = nc \log(cn + 1) + \log(cn + 1) + \log(cn + 1) - \log(cn + 1) \] (29.54)

\[ = (nc + 1) \log(cn + 1) + \log n + \log c + \log n + \log c - \log cn \] (29.55)

\[ = (nc + 1) \log cn + \log n + \log c + \log n + \log c - \log cn \] (29.56)
Main Theorem

... proof continued.

- And also by Lemma 29.5.5,

\[ H(U) \]
Main Theorem

... proof continued.

- And also by Lemma 29.5.5,

\[ H(U) \leq (EU + 1) \log(EU + 1) - EU \log EU \]  \hspace{1cm} (29.51)
Main Theorem

... proof continued.

And also by Lemma 29.5.5,

\[ H(U) \leq (EU + 1) \log(EU + 1) - EU \log EU \]  
\[ = \left( \frac{n}{c} + 1 \right) \log\left( \frac{n}{c} + 1 \right) - \frac{n}{c} \log \frac{n}{c} \]
Main Theorem

... proof continued.

And also by Lemma 29.5.5,

\[ H(U) \leq (EU + 1) \log(EU + 1) - EU \log EU \]  
\[ = (\frac{n}{c} + 1) \log(\frac{n}{c} + 1) - \frac{n}{c} \log \frac{n}{c} \]  
\[ = \frac{n}{c} \log(\frac{n}{c} + 1) + \log(\frac{n}{c} + 1) - \frac{n}{c} \log \frac{n}{c} \]  

(29.51)

(29.52)

(29.53)

(29.54)

(29.55)

(29.56)
... proof continued.

And also by Lemma 29.5.5,

\[
H(U) \leq (EU + 1) \log(EU + 1) - EU \log EU
\]  \hspace{1cm} (29.51)

\[
= \left( \frac{n}{c} + 1 \right) \log \left( \frac{n}{c} + 1 \right) - \frac{n}{c} \log \frac{n}{c}
\]  \hspace{1cm} (29.52)

\[
= \frac{n}{c} \log \left( \frac{n}{c} + 1 \right) + \log \left( \frac{n}{c} + 1 \right) - \frac{n}{c} \log \frac{n}{c}
\]  \hspace{1cm} (29.53)

\[
= \frac{n}{c} \log \frac{c}{n} \left( \frac{n}{c} + 1 \right) + \log \left( \frac{n}{c} + 1 \right)
\]  \hspace{1cm} (29.54)

(29.56)

...
And also by Lemma 29.5.5,

\[
H(U) \leq (EU + 1) \log(EU + 1) - EU \log EU
\]

\[
= (\frac{n}{c} + 1) \log(\frac{n}{c} + 1) - \frac{n}{c} \log \frac{n}{c}
\]

\[
= \frac{n}{c} \log(\frac{n}{c} + 1) + \log(\frac{n}{c} + 1) - \frac{n}{c} \log \frac{n}{c}
\]

\[
= \frac{n}{c} \log \left(\frac{n}{c} + 1\right) + \log(\frac{n}{c} + 1)
\]

(29.51)
Main Theorem

... proof continued.

And also by Lemma 29.5.5,

\[ H(U) \leq (EU + 1) \log(EU + 1) - EU \log EU \]  
(29.51)

\[ = \left( \frac{n}{c} + 1 \right) \log\left( \frac{n}{c} + 1 \right) - \frac{n}{c} \log \frac{n}{c} \]  
(29.52)

\[ = \frac{n}{c} \log \left( \frac{n}{c} + 1 \right) + \log \left( \frac{n}{c} + 1 \right) - \frac{n}{c} \log \frac{n}{c} \]  
(29.53)

\[ = \frac{n}{c} \log \left( \frac{n}{c} + 1 \right) + \log \left( \frac{n}{c} + 1 \right) \]  
(29.54)

\[ = \left( \frac{n}{c} + 1 \right) \log \left( \frac{n}{c} + 1 \right) + \log \frac{n+c}{c+n} \]  
(29.55)

\[ = \left( \frac{n}{c} + 1 \right) \log \left( \frac{n}{c} + 1 \right) + \log \frac{n+c}{c+n} \]  
(29.56)
Main Theorem

... proof continued.

And also by Lemma 29.5.5,

\[
H(U) \leq (EU + 1) \log(EU + 1) - EU \log EU
\]

\[
= \left(\frac{n}{c} + 1\right) \log\left(\frac{n}{c} + 1\right) - \frac{n}{c} \log \frac{n}{c}
\]

\[
= \frac{n}{c} \log\left(\frac{n}{c} + 1\right) + \log\left(\frac{n}{c} + 1\right) - \frac{n}{c} \log \frac{n}{c}
\]

\[
= \frac{n}{c} \log\left(\frac{n}{c} + 1\right) + \log\left(\frac{n}{c} + 1\right) + \log\left(\frac{c}{n} + 1\right) - \log\left(\frac{c}{n} + 1\right)
\]

\[
= \left(\frac{n}{c} + 1\right) \log\left(\frac{c}{n} + 1\right) + \log\frac{n+c}{c+n}
\]

\[
= \left(\frac{n}{c} + 1\right) \log\left(\frac{c}{n} + 1\right) + \log \frac{n}{c}
\]
... proof continued.

Thus, we have

$$c_n H(U, V) \leq c_n H(V) + c_n H(U) \leq c_n k + c_n \log n c + c_n (n c + 1) \log (c n + 1)$$

$$= c_n k + c_n \log n c + (c n + 1) \log (c n + 1)$$

$$= c_n k + c_n \log n c + \log (1 + c n) \rightarrow 0 \text{ as } n \rightarrow \infty$$

(29.61)
Thus, we have

\[ \frac{c}{n} H(U, V) \]
Thus, we have

\[
\frac{c}{n} H(U, V) \leq \frac{c}{n} H(V) + \frac{c}{n} H(U)
\]  

(29.57)
Thus, we have

\[
\frac{c}{n} H(U, V) \leq \frac{c}{n} H(V) + \frac{c}{n} H(U)
\]

\[
\leq \frac{c}{n} k + \frac{c}{n} \log \frac{n}{c} + \frac{c}{n} \left( \frac{n}{c} + 1 \right) \log \left( \frac{c}{n} + 1 \right)
\]  (29.57)

(29.58)
Thus, we have
\[c_n H(U, V) \leq c_n H(V) + c_n H(U)\] (29.57)
\[\leq c_n k + c_n \log n + \frac{c_n}{n} \log (1 + \frac{u}{c}) = c_n k + c_n \log n + \frac{c_n}{n} \log (1 + \frac{u}{c})\] (29.58)
\[(\Lambda) H_{\frac{u}{c}} + (\Lambda) H_{\frac{u}{c}} \geq (\Lambda, \Lambda) H_{\frac{u}{c}}\] (29.59)

\[
(1 + \frac{u}{c}) \log \left(1 + \frac{u}{c}\right) + \frac{c_n}{n} \log \left(1 + \frac{u}{c}\right) + \frac{c_n}{n} \log (1 + \frac{u}{c})
\]

\[
\rightarrow 0 \quad \text{as} \quad n \rightarrow \infty
\]

Therefore we have
Thus, we have

\[
\frac{c}{n} H(U, V) \leq \frac{c}{n} H(V) + \frac{c}{n} H(U) \tag{29.57}
\]

\[
\leq \frac{c}{n} k + \frac{c}{n} \log \frac{n}{c} + \frac{c}{n} \left( \frac{n}{c} + 1 \right) \log \left( \frac{c}{n} + 1 \right) \tag{29.58}
\]

\[
= \frac{c}{n} k + \frac{c}{n} \log \frac{n}{c} + \frac{c}{n} \left( \frac{n}{c} + 1 \right) \log \left( \frac{c}{n} + 1 \right) \tag{29.59}
\]

\[
= \frac{c}{n} k + \frac{c}{n} \log \frac{n}{c} + \frac{c}{n} \left( \frac{n}{c} + 1 \right) \log \left( \frac{c + n}{n} \right) \tag{29.60}
\]

\[
(29.61)
\]
Main Theorem

... proof continued.

Thus, we have

\[
\frac{c}{n} H(U, V) \leq \frac{c}{n} H(V) + \frac{c}{n} H(U) \tag{29.57}
\]

\[
\leq \frac{c}{n} k + \frac{c}{n} \log \frac{n}{c} + \frac{c}{n} \left( \frac{n}{c} + 1 \right) \log \left( \frac{c}{n} + 1 \right) \tag{29.58}
\]

\[
= \frac{c}{n} k + \frac{c}{n} \log \frac{n}{c} + \left( \frac{n}{c} + 1 \right) \log \left( \frac{c}{n} + 1 \right) \tag{29.59}
\]

\[
= \frac{c}{n} k + \frac{c}{n} \log \frac{n}{c} + \left( \frac{n}{c} + 1 \right) \log \left( \frac{c + n}{n} \right) \tag{29.60}
\]

\[
= \frac{c}{n} k + \frac{c}{n} \log \frac{n}{c} + \frac{c}{n} \log \left( 1 + \frac{c}{n} \right) + \log \left( 1 + \frac{c}{n} \right) \tag{29.61}
\]
Thus, we have

\[ \frac{c}{n} H(U, V) \leq \frac{c}{n} H(V) + \frac{c}{n} H(U) \tag{29.57} \]

\[ \leq \frac{c}{n} k + \frac{c}{n} \log \frac{n}{c} + \frac{c}{n} \left( \frac{n}{c} + 1 \right) \log \left( \frac{c}{n} + 1 \right) \tag{29.58} \]

\[ = \frac{c}{n} k + \frac{c}{n} \log \frac{n}{c} + \frac{c}{n} \left( \frac{n}{c} + 1 \right) \log \left( \frac{c}{n} + 1 \right) \tag{29.59} \]

\[ = \frac{c}{n} k + \frac{c}{n} \log \frac{n}{c} + \frac{c}{n} \log \left( 1 + \frac{c}{n} \right) \tag{29.60} \]

\[ \rightarrow 0 \text{ as } n \rightarrow \infty \]

\[ = \frac{c}{n} k + \frac{c}{n} \log \frac{n}{c} + \log \left( \frac{n}{c} + 1 \right) \tag{29.61} \]
... proof continued.

Thus, we have

\[
\frac{c}{n} H(U, V) \leq \frac{c}{n} H(V) + \frac{c}{n} H(U) \tag{29.57}
\]

\[
\leq \frac{c}{n} k + \frac{c}{n} \log \frac{n}{c} + \frac{c}{n} \left( \frac{n}{c} + 1 \right) \log \left( \frac{c}{n} + 1 \right) \tag{29.58}
\]

\[
= \frac{c}{n} k + \frac{c}{n} \log \frac{n}{c} + \left( \frac{n}{c} + 1 \right) \log \left( \frac{c}{n} + 1 \right) \tag{29.59}
\]

\[
= \frac{c}{n} k + \frac{c}{n} \log \frac{n}{c} + \frac{c}{n} \log \left( 1 + \frac{c}{n} \right) + \log \left( 1 + \frac{c}{n} \right) \tag{29.60}
\]

\[
\rightarrow 0 \text{ as } n \rightarrow \infty \quad \rightarrow 0 \text{ as } n \rightarrow \infty
\]
Thus, we have that
\[
\frac{c}{n} H(U, V) \leq \frac{c}{n} k + \frac{c}{n} \log \frac{n}{c} + o(1) \quad (29.62)
\]
\[
\text{→0 as } n \to \infty \quad \text{we'll look at this} \quad \text{→0}
\]
Main Theorem

... proof continued.

- Thus, we have that
  \[ \frac{c}{n} H(U, V) \leq \frac{c}{n} k + \frac{c}{n} \log \frac{n}{c} + o(1) \]  
  \( \rightarrow 0 \) as \( n \rightarrow \infty \)

- Now, by Lemma 29.5.3 we have
  \[ c(n) \leq \frac{n}{(1 - \epsilon_n) \log n} = \frac{n}{\log n} (1 + o(1)) < \frac{n}{c} \] for big enough \( n \)
Thus, we have that
\[
\frac{c}{n} H(U, V) \leq \frac{c}{n} k + \frac{c}{n} \log \frac{n}{c} + o(1) \quad (29.62)
\]
\[
\rightarrow 0 \quad \text{as} \ n \rightarrow \infty
\]

we'll look at this \(\rightarrow 0\)

Now, by Lemma 29.5.3 we have
\[
c(n) \leq \frac{n}{(1-\epsilon_n) \log n} = \frac{n}{\log n} (1 + o(1)) < \frac{n}{c} \quad \text{for big enough} \ n
\]

Then since \(c/n \log(n/c)\) is monotone up to its peak at \(n/c = e\),

\[
(29.65)
\]
Main Theorem

... proof continued.

- Thus, we have that
  \[
  \frac{c}{n} H(U, V) \leq \frac{c}{n} k + \frac{c}{n} \log \frac{n}{c} + o(1) \quad (29.62)
  \]
  \[\to 0 \text{ as } n \to \infty \quad \text{we'll look at this}\]

- Now, by Lemma 29.5.3 we have
  \[c(n) \leq \frac{n}{(1-\epsilon_n) \log n} = \frac{n}{\log n} (1 + o(1)) < \frac{n}{c} \text{ for big enough } n\]

- Then since \(c/n \log(n/c)\) is monotone up to its peak at \(n/c = e\),
  \[
  \frac{c}{n} \log \frac{n}{c}
  \]
  \[\leq O\left(\log \log \frac{n}{\log n}\right) \to 0 \text{ as } n \to \infty \quad (29.65)\]
Main Theorem

... proof continued.

Thus, we have that

\[
\frac{c}{n} H(U, V) \leq \frac{c}{n} k + \frac{c}{n} \log \frac{n}{c} + o(1) \quad \text{as } n \to \infty
\]

Now, by Lemma 29.5.3 we have

\[
c(n) \leq \frac{n}{(1-\epsilon_n) \log n} = \frac{n}{\log n} (1 + o(1)) < \frac{n}{c} \quad \text{for big enough } n
\]

Then since \( c/n \log(n/c) \) is monotone up to its peak at \( n/c = e \),

\[
\frac{c}{n} \log \frac{n}{c} \leq \frac{n}{\log n} (1 + o(1)) \log \frac{n}{\log n} (1 + o(1))
\]
Main Theorem

... proof continued.

- Thus, we have that

\[
\frac{c}{n} H(U, V) \leq \frac{c}{n} k + \frac{c}{n} \log \frac{n}{c} + o(1) \quad (29.62)
\]

\[
\to 0 \text{ as } n \to \infty \quad \text{we'll look at this} \quad \to 0
\]

- Now, by Lemma 29.5.3 we have

\[
c(n) \leq \frac{n}{(1 - \epsilon_n) \log n} = \frac{n}{\log n} (1 + o(1)) < \frac{n}{c} \quad \text{for big enough } n
\]

- Then since \( c/n \log(n/c) \) is monotone up to its peak at \( n/c = e \),

\[
\frac{c}{n} \log \frac{n}{c} \leq \frac{n}{\log n} (1 + o(1)) \log \frac{n}{\log n} (1 + o(1)) \quad (29.63)
\]

\[
= \log[\log n/(1 + o(1))] \frac{(1 + o(1))}{\log n} \quad (29.64)
\]

\[
(29.65)
\]
Main Theorem

... proof continued.

- Thus, we have that

\[
\frac{c}{n} H(U, V) \leq \frac{c}{n} k + \frac{c}{n} \log \frac{n}{c} + o(1) \quad (29.62)
\]

\[\rightarrow 0 \text{ as } n \rightarrow \infty\]

we'll look at this \[\rightarrow 0\]

- Now, by Lemma 29.5.3 we have

\[
c(n) \leq \frac{n}{(1 - \epsilon_n) \log n} = \frac{n}{\log n} (1 + o(1)) < \frac{n}{c} \text{ for big enough } n
\]

- Then since \(c/n \log(n/c)\) is monotone up to its peak at \(n/c = e\),

\[
\frac{c}{n} \log \frac{n}{c} \leq \frac{n}{\log n} (1 + o(1)) \log \frac{n}{\log n} (1 + o(1))
\]

\[\leq O\left(\frac{\log \log n}{\log n}\right) \rightarrow 0 \text{ as } n \rightarrow \infty\]
Thus, \( \frac{c}{n} H(U, V) \to 0 \) as \( n \to \infty \).
Main Theorem

... proof continued.

- Thus, \( \frac{c}{n} H(U, V) \rightarrow 0 \) as \( n \rightarrow \infty \).
- Therefore,

\[
\frac{c(n) \log c(n)}{n} \leq - \frac{1}{n} \log Q_k(x_{1:n}|s_1) + \epsilon_k(n) \tag{29.66}
\]

where \( \epsilon_k(n) \rightarrow 0 \) as \( n \rightarrow \infty \).
Thus, $\frac{c}{n}H(U, V) \to 0$ as $n \to \infty$.

Therefore,

$$\frac{c(n) \log c(n)}{n} \leq -\frac{1}{n} \log Q_k(x_{1:n}|s_1) + \epsilon_k(n)$$

(29.66)

where $\epsilon_k(n) \to 0$ as $n \to \infty$

Therefore,

$$\limsup_{n \to \infty} \frac{c(n) \log c(n)}{n} \leq \lim_{n \to \infty} -\frac{1}{n} Q_k(X_{1:n}|X_{-(k-1):0})$$

(29.67)

$$= H(X_0|X_{-1}, X_0, \ldots, X_k) \quad // \text{for stationary ergodic source}$$

$$\to H(X) \text{ as } k \to \infty$$

(29.68)
Theorem 29.5.8

Let $X_i$ be an infinite length stationary ergodic stochastic process. Let $\ell(x_{1:n})$ be the LZ codeword length for $n$ symbols. Then

$$\limsup_{n \to \infty} \frac{1}{n} \ell(x_{1:n}) \leq H(X)$$

(29.69)

Proof.

- We know that $\ell(x_{1:n}) = c(n)(\log(c(n)) + 1)$, where $c(n)$ is the number of phrases in the LZ parse (so they are distinct).
Theorem 29.5.8

Let $X_i$ be an infinite length stationary ergodic stochastic process. Let $\ell(x_{1:n})$ be the LZ codeword length for $n$ symbols. Then

$$\limsup_{n \to \infty} \frac{1}{n} \ell(x_{1:n}) \leq H(X)$$  \hspace{1cm} (29.69)

Proof.

- We know that $\ell(x_{1:n}) = c(n)(\log(c(n)) + 1)$, where $c(n)$ is the number of phrases in the LZ parse (so they are distinct).

- But from Lemma 29.5.3, we have

$$\limsup_{n \to \infty} \frac{c(n)}{n} = \limsup_{n \to \infty} \frac{1 + o(1)}{\log n} = 0$$ \hspace{1cm} (29.70)
Theorem 29.5.8

Let $X_i$ be an infinite length stationary ergodic stochastic process. Let $\ell(x_1:n)$ be the LZ codeword length for $n$ symbols. Then

$$\limsup_{n \to \infty} \frac{1}{n} \ell(x_1:n) \leq H(X)$$

(29.69)

Proof.

- We know that $\ell(x_1:n) = c(n)(\log(c(n)) + 1)$, where $c(n)$ is the number of phrases in the LZ parse (so they are distinct).

- But from Lemma 29.5.3, we have

$$\limsup_{n \to \infty} \frac{c(n)}{n} = \limsup_{n \to \infty} \frac{1 + o(1)}{\log n} = 0$$

(29.70)

Therefore,
Therefore,

\[
\limsup_{n \to \infty} \frac{\ell(x_1:n)}{n} = \limsup_{n \to \infty} \left( \frac{c(n) \log c(n)}{n} \to H(X) + \frac{c(n)}{n} \to 0 \right)
\]

(29.72)

\[
\leq H(X)
\]

(29.73)
Related to lengths

... proof continued.

Therefore,

\[
\limsup_{n \to \infty} \frac{\ell(x_1:n)}{n} = \limsup_{n \to \infty} \left( \frac{c(n) \log c(n)}{n} + \frac{c(n)}{n} \right) \rightarrow H(X)
\]

\[
\leq H(X)
\]

In other words, a purely algorithmic procedure (LZ), when faced with a (stationary ergodic) stochastic process governed by some distribution, but without knowing anything about the distribution and by only following the algorithm, will in the limit converge to the entropy rate of the stochastic process.