3 Self-Delimiting Kolmogorov complexity

3.1 Prefix codes

A set is called a prefix-free set (or a prefix set) if no string in the set is the proper prefix of another string in it. A prefix set cannot therefore be very "dense" in strings: if we take the first n strings from the standard enumeration, not many of those can be present in the same prefix set. A quantification of this is Kraft's inequality, which is a necessary condition for a prefix set.

Lemma 3.1.1. *(Kraft) Let A be any prefix set. Then*

$$\sum_{x \in A} \frac{1}{2^{|x|}} \leq 1.$$

We give a probabilistic proof of the lemma.

Proof. Consider the toss of a fair coin, with identical probability (0.5) for both the outcomes. We consider the random experiment of tossing the coin for a number of times until the string of outcomes x is a member of A. Let us denote the event as E_x. Since A is a prefix set, if x occurs in A, no extension of x will be in A. Thus,

$$P(E_x) = 2^{-x}.$$

Since A being a prefix code implies that E_x and E_y are mutually disjoint for distinct x and y, and

$$\cup_{x \in A} E_x = A,$$

we have

$$P(A) = \sum_{x \in A} P(E_x) = \sum_{x \in A} \frac{1}{2^{|x|}} \leq 1,$$

where the last inequality is true because $P(A)$ is a probability. □

A bijection $f : \Sigma^* \to A$ is called a *prefix code* if A is a prefix set.

Kraft's inequality is not a sufficient condition for prefix sets. There is a general result, called *McMillan's Inequality* which proves that the above inequality characterizes *uniquely decodable codes* - these are a superset of prefix codes. We omit its proof.

Kraft's inequality will be useful in developing algorithmic probability, later in the course.

3.2 Self-delimiting Kolmogorov complexity

One of the undesirable properties of plain Kolmogorov complexity was its subadditivity. Self-delimiting Kolmogorov complexity was one of the approaches introduced to address this issue, and it solves most of the defects of plain Kolmogorov complexity except the nonmonotonicity over prefixes.

The main idea is to restrict the set of programs to be a prefix set. We have to redo some of the work that we did for plain Kolmogorov complexity here, since it is not clear immediately that the definition will be independent of the UTM we choose.

Let $\phi_1, \phi_2, \ldots$ be the computable enumeration of all the unary partial computable functions mapping strings to strings, and let $T_1, T_2, \ldots$ be the corresponding Turing machines.

We will produce a new computable enumeration $\phi'_1, \phi'_2, \ldots$ of unary partial computable *prefix* functions - that is, the domain of each of these functions is a prefix set.

Consider a Turing machine T_i which computes a partial computable function ϕ_i. We want to define a new Turing machine T'_i which computes over a prefix set, such that if ϕ_i is a partial computable prefix function, then $\phi_i = \phi'_i$. Given an input $b_0 b_1 \ldots b_{m-1}$, T'_i operates in the following manner (It is important to note that T'_i does not read any input bit to start with.)

1.Set $p = \lambda$, and $m = -1$.
2.Dovetail over all strings q, the execution of T_i over pq. Let q' be the first string in the dovetailed execution such that T_i halts on (p, q'). If such a q' is found, goto 3.
3.If $q' = \lambda$, output $\phi_i(p)$. Otherwise, increment m by 1, and set $p = b_0 \ldots b_m$
4.go to 2.

The above code ensures that the domain of definition of T'_i is a prefix set. If ϕ_i was a partial computable prefix function, then so is ϕ'_i. [1] Thus, there is a computable enumeration $\phi'_1, \phi'_2, \ldots$ of partial computable prefix functions, and hence of their corresponding Turing machines $T'_1, T'_2, \ldots$. This is enough to prove that there is a universal partial computable prefix function for which an invariance theorem holds.

Recall that the plain Kolmogorov complexity of x conditioned on y, with respect to a partial computable function $\phi : \Sigma^* \to \Sigma^*$ is

$$C_\phi(x|y) = \min\{|p| : \phi(\llbracket y, p \rrbracket) = x\}.$$

Encoding: Let us consider some fixed prefix encoding $e : \Sigma^* \to A$, where A is a prefix set. Then $\langle x, y \rangle$ defined by concatenating $e(x)$ with $e(y)$ is a prefix encoding of pairs of strings (x, y). Thus $|\langle x, y \rangle| = |e(x)| + |e(y)|$. [2]

[1] T'_i may not halt if there is no suffix y of some prefix $b_0 b_1 \ldots b_{k-1}$ of the given input, on which T_i halts - so in general T'_i is partial computable.

[2] Contrast with the blowup for a general uniquely decodable code: for example, $|\llbracket x, y \rrbracket| = 2|x| + |y| + 2$.

HW2(a) asks you to define a prefix encoding for pairs of strings. We will assume that it is possible to define an encoding e such that $|e(x)| \leq 2\log|x| + 2 + |x|$. Thus, $|\langle x, y\rangle| \leq |x| + |y| + 2(\log|x| + \log|y| + 2)$.

Theorem 3.2.1. *There is a universal partial computable prefix function $\psi : \Sigma^* \to \Sigma^*$ such that for any partial computable prefix function $\phi : \Sigma^* \to \Sigma^*$ there is a constant c_ϕ for which the following holds. For any strings x and y,*

$$C_\psi(x|y) \leq C_\phi(x|y) + c_\phi$$

Proof. We can construct a universal prefix Turing machine U, which, given input $\langle y, \langle s_n, p\rangle\rangle$ computes $T_n'(\langle y, p\rangle)$. Let $\psi : \Sigma^* \to \Sigma^*$ be the function computed by U. Then by an argument similar to the invariance theorem for plain Kolmogorov complexity, it follows that

$$C_\psi(x|y) \leq C_{\phi_n'}(x|y) + c_{\phi_n'},$$

where $c_{\phi_n'} = 2\log|s_n| + |s_n| + 2$. □

Thus, we can define:

Definition 3.2.2. The *prefix complexity* of x given y, denoted $K(x|y)$ is defined to be $C_\psi(x|y)$ for all strings x and y. The *unconditional prefix complexity* of x, denoted $K(x)$, is $K(x|\lambda)$.

(1) K is now subadditive. To see this, let x and y be two strings, and let p_x be a shortest prefix program for x and p_y be a shortest prefix program for y. We know that there is a universal prefix Turing machine U such that $U(p_x) = x$ and $U(p_y) = y$. Then, we can construct a prefix Turing machine V such that, given input $\langle p_1, p_2\rangle$, it computes $(U(p_1), U(p_2))$. Thus, we have

$$K(x, y) \;\leq\; K_V(x, y) + O(1) \;\leq\; K(x) + K(y) + O(1). \tag{1}$$

(2) We also have the following relation between C and K.

$$C(x|y) \leq K(x|y) + O(1).$$
$$K(x|y) \leq C(x|y) + 2\log C(x|y) + O(1).$$

The first inequality follows directly from the fact that the universal partial computable prefix function is a partial computable function. The second inequality follows from the following argument. Recall that $C(x|y) = \min\{|p| : \phi(\llbracket y, p\rrbracket) = x\}$, where ϕ is the universal partial computable function we chose. There is an index n in the computable enumeration of partial computable functions at which ϕ occurs. That is, $\phi = \phi_n$. Then, the universal *prefix* Turing machine U, given $\langle y, s_n, p\rangle$ can compute $T_n(\llbracket y, p\rrbracket) = \phi_n(\llbracket y, p\rrbracket)$. [3] Then, it follows that

$$\begin{aligned} K(x|y) &\leq |e(s_n)| + |e(p)| + O(1) \\ &\leq 2\log(|s_n|) + |s_n| + 2\log C(x|y) + C(x|y) + O(1) \\ &\leq 2\log C(x|y) + C(x|y) + O(1). \end{aligned}$$

[3] Note: We are not computing $T_n'(\langle y, p\rangle)$!.

Lemma 3.2.3. *1. For each n, the maximal complexity of n length strings is $n + K(n) + O(1)$.*

2. For each r, the number of strings of length n with

$$K(x) \leq n + K(n) - r$$

is at most $2^{n-r+O(1)}$.

Proof. We prove the second proposition first. Let r be an arbitrary number. Assume that a string x of length n satisfies

$$K(x) \leq n + K(n) - r. \tag{2}$$

Then we have (IOU)

$$K(x) + K(n \mid x, K(x)) = K(n) + K(x \mid n, K(n)) + O(1).$$

Since x is an n-length string,

$$K(n|x, K(x)) = O(1).$$

So

$$K(x) + O(1) = K(n) + K(x \mid n, K(n)) + O(1) \tag{3}$$

Then (2) and (3) gives

$$K(x|n, K(n)) \leq n - r + O(1),$$

from which we can conclude that the number of strings satisfying (2) is at most $2^{n-r+O(1)}$.

This gives the lower bound for the first proposition. □

3.3 Coding Theorem

[4] If we have a set of prefix Turing machines $\mathcal{M} = \{p_0, p_1, \dots\}$ encoded in binary, then the Kraft inequality gives us

$$\sum_{i=1}^{\infty} \frac{1}{2^{|p_i|}} \leq 1.$$

This allows us to define the following notion.

[4] This section has been prepared from notes prepared by David Doty.

Definition 3.3.1. Let $x \in \{0,1\}^*$. The *algorithmic probability* of x is

$$\mathbf{m}(x) = \sum_{\substack{p \in \mathcal{M} \\ U(p)=x}} \frac{1}{2^{|p|}}.$$

We can verify that $\sum_{x \in \{0,1\}^*} \mathbf{m}(x) \leq 1$. (Technically, $\mathbf{m}$ is a subprobability.) Thus $\mathbf{m}(x)$ is the probability that U will output x when its input bits are determined by independent tosses of a fair coin.

We now try to show that $K(x) = -\log \mathbf{m}(x) + O(1)$, thus developing the analogy between K and the Shannon entropy $\mathcal{H}$. It is clear that $K(x) \geq -\log \mathbf{m}(x)$, since $2^{-K(x)}$ is one of the terms that constitute the sum defining $\mathbf{m}(x)$. The converse inequality is called Levin's Coding Theorem.

We first introduce some definitions needed for the discussion.

Definition 3.3.2. An *indexed enumerator* is a Turing machine M taking an input $z \in \{0,1\}^*$ such that $M(z)$ enumerates a computably enumerable language.

We denote the language enumerated by Turing machine M by $L(M)$. Thus, $M(\lambda), M(0), M(1), \ldots$ is an enumeration of (not necessarily all) Turing machines, and the corresponding computably enumerable languages are $L(M(\lambda)), L(M(0)), ...$

Definition 3.3.3. A *request set* is a set L of pairs of binary strings (x, s_n) such that

$$\sum_{(x,s_n) \in L} \frac{1}{2^n} \leq 1.$$

We call each (x, s_n) a *request*, and n its *request length*.

Definition 3.3.4. A *conditional requester* is an indexed enumerator R such that, for each $z \in \{0,1\}^*$, $R(z)$ enumerates a request set.

Thus R is a Turing machine, which, given a binary string z as input, will output the *Turing machine* enumerating language $R(z)$. That is,

$$L(R(\lambda)), L(R(0)), L(R(1)), \ldots$$

is a computably enumerable sequence of request sets.

The following theorem shows that if a request set $L(R(z))$ is computably enumerable, then if $(x, s_n) \in L(R(z))$, the complexity of x conditioned on z is at most the request length of the request. We prove this with the help of an effective converse of Kraft's inequality.

That is, given a computable enumeration of lengths $n_0, n_1, n_2, \ldots$ satisfying $\sum_{i=1}^{\infty} \frac{1}{2^i} \leq 1$, we can computably enumerate a prefix set of codewords $c_0, c_1, \ldots$ such that for all $i \in \mathbb{N}$, $|c_i| = n_i$.

Theorem 3.3.5. *For each conditional requester R, there exists a constant c_R such that, for any binary string z, and all $(x, s_n) \in L(R(z))$,*

$$K(x|z) \leq n + c_R.$$

Proof. Suppose that on getting a binary string z as input, $R(z)$ enumerates $(x_0, s_{n_0}), (x_1, s_{n_1}), \ldots$. We will assign a codeword c_i from a *prefix set*, to each request (x_i, s_{n_i}) such that $|c_i| = n_i$ and c_i represents x_i.

We describe a Turing machine M that, on input c_i and z, will output x_i. If the input c is not a codeword, then the computation diverges. Let the program size of this machine be l_M. This machine is sufficient to show that $K(x_i|z) \leq |c_i| + l_M = n_i + l_M$.

$M(c_i, z)$
1. $C = \emptyset$. ▷ *The set of codewords*
2. $j = 0$. ▷ *index for the language $L(R(z))$.*
3. **for** each (x_j, s_{n_j}) enumerated by $R(z)$ **do**
4. $w =$ the first element in $\{0,1\}^{n_j} \setminus C$ with no proper prefix or extension already in C.
5. **if** $w = c_i$ **then**
6. output x_j and halt. ▷ *c_i is the encoding of (x_j, s_{n_j})*
7. **else** $C[j] = w$, increment j by 1. ▷ *Add w to the set of assigned code words.*
8. **done**

We will now prove that the algorithm works as we claim. That is, it defines a prefix encoding of the sequence $(x_1, s_{n_1}), (x_2, s_{n_2}), \ldots$ such that the codeword c_i assigned to (x_i, s_{n_i}) is n_i bits long.

We have to show that it is always possible to choose a string w in line 4. That is, if (x, s_n) is enumerated by the machine $R(z)$, then there is a string w with n bits such that no prefix or extension of w has already been assigned by the algorithm in its iteration thus far.

Let us denote the set of infinite binary sequences with the string w as prefix by $[w]$. Let w be the k^{th} string of length n. By the binary representation of the unit interval $[0, 1]$, this corresponds to the interval

$$\left[\frac{k}{2^n}, \frac{k}{2^n}\right).$$

The set C at any point in the algorithm is a finite union of such intervals. Since we ensure that when we add any string w to the set C, no prefix or extension of w is already in C, we can observe that C is the finite disjoint union of such intervals. Also, at any iteration j of the algorithm,

$$\frac{1}{2^{|C[0]|}} + \frac{1}{2^{|C[1]|}} + \cdots + \frac{1}{2^{|C[j]|}} = \frac{1}{2^{s_{n_1}}} + \frac{1}{2^{s_{n_2}}} + \cdots + \frac{1}{2^{s_{n_j}}}. \tag{4}$$

We now claim that the algorithm ensures that the complement of C, denoted C^c, can be written as a finite disjoint union of intervals with length strictly increasing to the right. That is,

$$[C^c] = \bigcup_{i=1}^{m} \left[\frac{k_i}{2^{n_i}}, \frac{k_{i+1}}{2^{n_i}} \right), \text{ where } n_i > n_j \text{ if } 1 \leq i < j \leq m. \tag{5}$$

Proof of this claim can be found here.

Now, let us assume that every $w \in \{0,1\}^n$ has a prefix or extension already in C. This means that every subinterval in the disjoint union (5) has length at most 2^{-n-1}. By the claim above, the length of $[C^c]$ is at most

$$\frac{1}{2^{n+1}} + \frac{1}{2^{n+2}} + \frac{1}{2^{n+3}} + \dots,$$

which is equal to $\frac{1}{2^n}$. Thus C has measure greater than $1 - \frac{1}{2^n}$. By the observation (4), this means that no more request with request length n will ever be issued. □

Suppose we consider a machine which, for any input w, enumerates the language $R(z)$. Then on input λ too, the machine will enumerate $R(z)$. Thus, we have the unconditional complexity version of the above theorem.

Corollary 3.3.6. *If L is a computably enumerable request set, then there is a constant c_L such that for all $(x, s_n) \in L$,*

$$K(x) \leq n + c_L.$$

The main theorem of this subsection intuitively states that if there are many long programs which output a string x, then there is a short program which outputs it. This result is known as the Coding Theorem.

Theorem 3.3.7 (Coding Theorem (Levin))**.** *There is a constant c_{coding} such that for all strings x,*

$$K(x) \leq -\log \mathbf{m}(x) + c_{coding}.$$

The intuition of its proof is as follows. We will consider a set of requests (x, s_n) with every s_n at least as long as $-\log \mathbf{m}(x)$. If this set is computably enumerable, then Corollary 3.3.6 assures us that $K(x)$ is at most $-\log \mathbf{m}(x) + O(1)$.[5]

Proof. Define the set

$$L = \{(x, s_n) \mid -\log \mathbf{m}(x) < n + 1\}.$$

It suffices to show that L is a computably enumerable request set. For, then by Theorem 3.3.5, we can construct a prefix set where each x corresponds to a program (codeword) with length n.

[5] A subtlety of the proof is this: $\mathbf{m}(x)$ is not computable - so we cannot ask the precise question $(x, s_{-\log \mathbf{m}(x)+1})$ alone. We will have to ask a lot more questions (x, s_n) where $-\log \mathbf{m}(x) < n+1$ because $\mathbf{m}(x)$ is known only "in the limit." This is why we needed an effective converse of Kraft's inequality.

First, we prove that L is computably enumerable. Recall that L is computably enumerable if and only if there is a Turing machine which accepts it. We design a machine M that accepts (x, s_n) exactly when $-\log \mathbf{m}(x) < n+1$. This machine, on input (x, s_n) starts dovetailing over all prefix Turing machines, computing a lower approximation $\underline{\mathbf{m}}(x)$ of $\mathbf{m}(x)$. Initially, M sets $\underline{\mathbf{m}}(x) = 0$. If the prefix machine p_i halts and outputs x, then M adds $\frac{1}{2^{|p_i|}}$ to $\underline{\mathbf{m}}(x)$. This continues until M can ensure that $-\log \underline{\mathbf{m}}(x) < n+1$, when it accepts (x, s_n). If this condition is not met, M runs forever.

This algorithm will accept exactly those $(x, s_n) \in L$: $\mathbf{m}$ is approximable from below, and in the limit, $\underline{\mathbf{m}}(x)$ equals $\mathbf{m}(x)$. Thus $-\log \mathbf{m}(x)$ is approximable from above, and in the limit, $-\log \underline{\mathbf{m}}(x)$ equals $-\log \mathbf{m}(x)$. Thus, if $n+1$ is greater than $-\log \mathbf{m}(x)$, then at some point, it will be greater than $-\log \underline{\mathbf{m}}(x)$, and M will accept.

Thus L is computably enumerable.

We also have

$$\begin{aligned}\sum_{(x,s_n)\in L} \frac{1}{2^{|s_n|}} &= \sum_{x\in\Sigma^*} \sum_{-\log \mathbf{m}(x)-1<n} \frac{1}{2^n} \\ &\leq \sum_{x\in\Sigma^*} \mathbf{m}(x) \\ &\leq 1,\end{aligned}$$

proving that L is a request set.

□

3.4 Symmetry of Information

The theorem in this section shows that, up to a small error term, $K(x, y)$ (the amount of randomness in the pair (x, y)) is the sum of $K(x)$ (the amount of randomness in x), and $K(y|x)$ (the amount of randomness in y even when we are given access to x).

The following lemma states that the algorithmic probability of any string x is lower bounded by the probability over all strings y of the pair (x, y).

Lemma 3.4.1. *There is a constant $c_{Projection}$ such that for all $x \in \Sigma^*$,*

$$\mathbf{m}(x) \geq \frac{1}{2^{c_{Projection}}} \sum_{\substack{y,p\in\Sigma^* \\ U(p)=(x,y)}} \frac{1}{2^{|p|}}.$$

Proof. Let x, y be binary strings, and let p be the binary encoding of a prefix program such that $U(p) = (x, y)$. Then define q_p to be the program that simulates p to produce (x, y) and then outputs x. Then $U(q_p) = x$ and there is a constant $c_{\text{Projection}}$ independent of x (actually, even independent of p) such that $|q_p| \leq |p| + c_{\text{Projection}}$.

Then

$$\begin{aligned}\mathbf{m}(x) &= \sum_{\substack{q\in\Sigma^* \\ U(q)=x}} \frac{1}{2^{|q|}} \\ &\geq \sum_{\substack{y,p\in\Sigma^* \\ U(p)=(x,y)}} \frac{1}{2^{|q_p|}} \\ &\geq \frac{1}{2^{c_{\text{Projection}}}} \sum_{\substack{y,p\in\Sigma^* \\ U(p)=(x,y)}} \frac{1}{2^{|p|}}\end{aligned}$$

□

We know that

$$K(x,y) \leq K(x) + K(y|x) + O(1),$$

thus

$$K(x,y) \leq K(x) + K(y|x, K(x)) + O(1).$$

The following nearly gives the converse inequality[6] which is the difficult direction in symmetry of information.

Theorem 3.4.2. *There is a constant $c \in \mathbb{N}$ such that for all $x, y \in \Sigma^*$,*

$$K(y|x, K(x)) \leq K(x,y) - K(x) + c.$$

Proof. Let R be the Turing machine, which, when given input $z = \langle x, s_n\rangle$ does the following. If there is a program of length exactly n which produces x[7], then $R(z)$ enumerates the language

$$L(R(z)) = \{\langle y, |p| - n + c_{\text{coding}} + c_{\text{projection}}\rangle \mid U(p) = (x,y)\}.$$

Then, $L(R(z))$ is computably enumerable: The machine verifies that the input is of the form $\langle x, s_n\rangle$, and if so, dovetails over all programs of length n. If any of them halt and produce x, then $R(z)$ enumerates $L(R(z))$.

When the string $z' = \langle x, K(x)\rangle$ is input, we have that

$$L(R(z')) = \{\langle y, |p| - K(x) + c_{\text{coding}} + c_{\text{projection}}\rangle \mid U(p) = (x,y)\}.$$

Thus $L(R(z'))$ is computably enumerable.

[6]The exact converse of the inequality in Theorem 3.4.2 is

$$K(x,y) \leq K(x) + K(y|x, K(x)) - c$$

for some $c \in \mathbb{N}$, but we have this inequality only up to an additive constant.

[7]There could be more than one of the same length.

Now, we show that $L(R(z'))$ is a request set. Even though it is not necessary that every $L(R(z))$ is a request set, on the particular input z', we have the following inequality,

$$\begin{aligned}
\sum_{\langle y,m\rangle \in L(R(z'))} \frac{1}{2^m} &= \sum_{\substack{y,p\in\Sigma^* \\ U(p)=(x,y)}} \frac{1}{2^{|p|-K(n)+c_{\text{coding}}+c_{\text{projection}}}} \\
&= 2^{K(x)-c_{\text{coding}}-c_{\text{projection}}} \sum_{\substack{y,p\in\Sigma^* \\ U(p)=x,y}} \frac{1}{2^{|p|}} \\
&\leq 2^{K(x)-c_{\text{coding}}-c_{\text{projection}}} \quad 2^{c_{\text{projection}}}\mathbf{m}(x) && \text{by Lemma 3.4.1} \\
&\leq 2^{K(x)-c_{\text{coding}}} 2^{c_{\text{coding}}} \frac{1}{2^{K(x)}} && \text{by Theorem 3.3.7} \\
&= 1,
\end{aligned}$$

which proves that $L(R(z'))$ is a request set.

By Theorem 3.3.5, for all $x, y \in \Sigma^*$,

$$K(y|x, K(x)) \leq |p| - K(x) + c_{\text{coding}} + c_{\text{projection}} + c_R,$$

for any program p such that $U(p) = (x, y)$. In particular, for the case when p is a minimal program for (x, y), we have

$$K(y|x, K(x)) \leq K(x, y) - K(x) + c.$$

where $c = c_{\text{coding}} + c_{\text{projection}} + c_R$. □

With this, we can now prove the following estimate which helps to prove the symmetry of information. [8]

Corollary 3.4.3. *There is a constant $c' \in \mathbb{N}$ such that for all strings x and y,*

$$K(y|x) \leq K(x, y) - K(x) + K(K(x)) + c'.$$

Proof. Let M be the minimal program from Theorem 3.4.3 which testifies that

$$K(y|x, K(x)) \leq K(x, y) - K(x) + K(K(x)) + c.$$

Let $p_{K(x)}$ be a minimal program for $K(x)$. Then a program with $p_{K(x)}$ hardcoded, and on input x outputs the value of $M(x, K(x))$ is not more than $|M| + |p_{K(x)}| + O(1)$ bits long, thus we have

$$K(y|x) \leq K(x, y) - K(x) + K(K(x)) + c'.$$

□

[8] Thanks to Anuradha Mehta, Mrinalkanti Ghosh, Santosh Kumar, Rohit Gurjar and Rahul Gupta for pointing out various problems with the original formulation of Theorem 3.4.2 and Corollary 3.4.3.

Note that

$$K(x,y) \leq K(y,x) + O(1),$$

since there is a program that merely inverts the order of outputs of a machine which prints (y,x).

Thus, by Corollary 3.4.3 and the inequality (1) we have the following.

$$K(x) + K(y|x) - K(K(x)) + O(1) \ \leq \ K(x,y) \ \leq \ K(x) + K(y|x) + O(1).$$

Overestimating the quantity on the right side, we have the more uniform

$$K(x) + K(y|x) - K(K(x)) + O(1) \ \leq \ K(x,y) \ \leq \ K(x) + K(y|x) + K(K(x)) + O(1).$$

from which it follows that

$$|[K(x) + K(y|x)] - K(x,y)| \leq K(K(x)) + O(1).$$

Similarly,

$$|K(y,x) - [K(y) + K(x|x)]| \leq K(K(y)) + O(1).$$

Adding the above two inequalities and rearranging, we have

$$|[K(x) - K(x|y)] - [K(y) - K(y|x)]| \qquad \leq \quad K(K(x)) + K(K(y)) + O(1).$$

We can define mututal information of x and y as

$$I(x:y) = K(x) - K(x|y).$$

This is the amount of information in x that can be obtained from y. The inequality above can be rephrased as

$$|I(x:y) - I(y:x)| \leq K(K(x)) + K(K(y)) + O(1).$$

which we call the *weak symmetry of information*.

If we replace x by $\langle x, K(x)\rangle$ in Theorem 3.4.2, then we get

$$K\big(\langle x, K(x)\rangle, y\big) = K\big(\langle x, K(x)\rangle\big) + K\big(y|\langle x, K(x)\rangle\big) + O(1).$$

Then we can see that *symmetry of information* holds in the following form.

$$I(\langle x, K(x)\rangle : y) = I(\langle y, K(y)\rangle : x) + O(1).$$

Symmetry of information is one of the crucial features of Kolmgorov complexity - we give an estimate for the difference in mutual information between *any* two strings. We will see that symmetry of information may cease to hold when we restrict the notion of Kolmogorov complexity to *resource-bounded* Turing machines. For example, it is likely that under some plausible hypotheses in cryptography, like the existence of one-way functions, there is no symmetry of information for polynomial-time Turing machines. We will discuss this later in the course.