CS 687 Midsem

Rohan (241110057)

1 Suppose $f : \Sigma^* \to \Sigma^*$ is one-to-one (i.e. an injective function) and is partial computable. Show that f^{-1} is partial computable - i.e. there is a Turing machine $M : \Sigma^* \to \Sigma^*$ such that for every $y \in range(f), M(y)$ halts and outputs the a value x such that $f(x) = y$.

Solution:

Understanding Partial Computability

A function $f : \Sigma^* \to \Sigma^*$ is partial computable if there exists a Turing machine M_f that computes $f(x)$ for all $x \in \text{domain}(f)$ but may not halt for $x \notin \text{domain}(f)$.
Since f is injective, so for every $y \in \text{range}(f)$ there exists a unique x such that $f(x) = y$.

Constructing a Turing Machine for f^{-1}

Since f is partial computable we can enumerate all pairs $(x, f(x))$ using a universal Turing machine that simulates f on all x.

To construct M, the Turing machine computing f^{-1}, we follow these steps:

1. **Enumerate Inputs:** Generate an enumeration of all possible inputs $x \in \Sigma^*$.

2. **Dovetailing:** Use a dovetailing technique to simulate $f(x)$ for all x in parallel. Dovetailing works by interleaving steps of computation for multiple inputs. For example:

   - Step 1: Compute $f(x_1)$ for 1 step.

- Step 2: Compute $f(x_1)$ for 2 steps and $f(x_2)$ for 1 step.
- Step 3: Compute $f(x_1)$ for 3 steps, $f(x_2)$ for 2 steps, and $f(x_3)$ for 1 step.
- Continue this process indefinitely.

3. **Compare Outputs:** Whenever $f(x)$ halts and produces an output $f(x)$, compare it to the given input y.
   - If $f(x) = y$, output x and halt.
   - If $y \notin \text{range}(f)$, the machine M will not halt.

Correctness Proof

- Since f is injective, each $y \in \text{range}(f)$ has exactly one corresponding x such that $f(x) = y$.
- The dovetailing technique ensures that M will eventually find this x if it exists.
- If $y \notin \text{range}(f)$ M will not halt which is consistent with the definition of a partial computable function.

Example

Suppose f is defined as follows:

- $f(0) = 00$
- $f(1) = 01$
- $f(10) = 10$

Given $y = 01$, the machine M will:

1. Compute $f(0) = 00$ (not equal to y).
2. Compute $f(1) = 01$ (equal to y), so M outputs 1 and halts.

Conclusion

We have explicitly constructed a Turing machine M that computes $f^{-1}(y)$ for all $y \in \text{range}(f)$. Since M halts only for $y \in \text{range}(f)$, f^{-1} is partial computable.

2 Consider a string x with $K(x) \geq |x| - c$. Let y be an arbitrary string. Show that $|x| + K(y|x) \leq K(x, y) + O(1)$. (In words: if you condition on an incompressible string, the symmetry of information inequality is much simpler.)

Solution:

Using the chain rule

Using the Kolmogorov complexity chain rule:

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

Since x is incompressible:

$$K(x) \geq |x| - c$$

Substituting this into the inequality gives:

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

Rearranging:

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

Explanation of the Inequality

- The term $|x|$ represents the length of x.
- $K(y|x)$ is the conditional Kolmogorov complexity of y given x.
- The inequality shows that when x is incompressible the sum $|x| + K(y|x)$ is bounded by $K(x, y) + O(1)$.

Example

Let $x = 000\ldots0$ (a string of n zeros) and $y = 111\ldots1$ (a string of n ones). Since x is highly structured, $K(x) \approx \log n$. If y is random, $K(y|x) \approx n$. Then:

$$|x| + K(y|x) \approx n + n = 2n$$

and

$$K(x, y) \approx n + n = 2n$$

Thus, the inequality holds.

Conclusion

The inequality $|x| + K(y|x) \leq K(x, y) + O(1)$ holds for incompressible x, simplifying the symmetry of information inequality.

3 Suppose, for every $k > 1$, you can obtain the first k bits of Chaitin's Omega (described in Question 7 of Homework 1). Using this, define an algorithm to decide whether any program $p \in P$ of length $< k$ halts, where P is the prefix-free set of programs.

Solution:

Understanding Chaitin's Omega

Chaitin's Omega is defined as:

$$\omega = \sum_{p \in P, M(p)\downarrow} 2^{-|p|}$$

where P is a prefix-free set of programs and $M(p)$ is a universal prefix-free Turing machine. The first k bits of ω encode information about the halting status of programs of length $< k$.

Algorithm to Decide Halting

Given the first k bits of ω we can decide whether a program $p \in P$ of length $< k$ halts as follows:

1. **Enumerate Programs:** Enumerate all programs $p \in P$ of length $< k$.

2. **Compute Contributions:** For each program p compute its contribution $2^{-|p|}$ if $M(p)$ halts.

3. **Compare to ω:** Sum the contributions of all halting programs of length $< k$. Compare this sum to the first k bits of ω.

4. **Decision:** If the sum matches the first k bits of ω, all programs of length $< k$ halt.
   Otherwise at least one program does not halt.

Example

Suppose $k = 3$, and the first 3 bits of ω are 0.101. We enumerate all programs of length < 3:

- $p_1 = 0$ (length 1)
- $p_2 = 1$ (length 1)
- $p_3 = 00$ (length 2)
- $p_4 = 01$ (length 2)
- $p_5 = 10$ (length 2)
- $p_6 = 11$ (length 2)

If the sum of contributions from halting programs matches 0.101, all programs of length < 3 halt.

Conclusion

Given the first k bits of ω, we can decide the Halting Problem for programs of length $< k$. This does not contradict the general undecidability of the Halting Problem because it only works for programs of bounded length.

4 Show that for every pair of strings x and y, we have $C(x, y) \leq C(x) + C(y) + \log C(x) + \log C(y) + O(1)$.

Solution:

Using the Chain Rule

By the Kolmogorov complexity chain rule:

$$C(x, y) = C(x) + C(y|x) + O(1)$$

Since conditioned complexity is at most total complexity,

$$C(y|x) \leq C(y)$$

Thus:

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

Refining the Bound

To encode both $C(x)$ and $C(y)$, we need an additional $\log C(x) + \log C(y) + O(1)$ bits. This is because:

- The length of $C(x)$ is $\log C(x)$.
- The length of $C(y)$ is $\log C(y)$.

Thus, the refined bound is:

$$C(x, y) \leq C(x) + C(y) + \log C(x) + \log C(y) + O(1)$$

Example

Let $x = 000\ldots0$ (a string of n zeros) and $y = 111\ldots1$ (a string of n ones). Then:

$$C(x) \approx \log n, \quad C(y) \approx \log n$$

The bound becomes:

$$C(x, y) \leq \log n + \log n + \log\log n + \log\log n + O(1)$$

Conclusion

The bound $C(x, y) \leq C(x) + C(y) + \log C(x) + \log C(y) + O(1)$ holds, showing that encoding both $C(x)$ and $C(y)$ incurs an additional logarithmic overhead.