7 Computability

Computer Science Mentors 70

Mar 08 - Mar 12
Spring 2021

Amogh Gupta, Sylvia Jin, Aekus Bhathal, Abinav Routhu, Debayan Bandyopadhyay
Roast us here: `https://tinyurl.com/csm70-feedback20`

Computability

1. Say that we have a program M that decides whether any input program halts as long as it prints out the string "ABC" as the first operation that it carries out. Can such a program exist? Prove your answer.

Solution: No. Such a program M can not exist. We proceed as follows: we show that if such a program existed, the halting problem would be computable.

Consider any program P. If we wanted to decide if P halted, we could simply create a new program P' where P' first prints out "ABC", then proceed to do exactly what P would. However, if M existed, we could determine whether any program P halted by converting it to a P' and feeding it into M. This would solve the halting problem - but this is a contradiction, since by diagonalization we can prove that the halting problem is not computable.

Therefore, M can not exist!

Below we have the Pseudocode for our program P.

```
define M'(Program p, input i):
    Construct program p' which will first print('ABC')
        then run p(i)
    run M(p', i)

define M(Program p, input i):
    if p's first command is print('ABC'):
        if p(i) halts:
            return True
        else if p(i) loops forever:
            return False
```

2. (a) Is it possible to write a program that takes a natural number n as input, and finds the shortest arithmetic formula which computes n? For the purpose of this question, a formula is a sequence consisting of some valid combination of (decimal) digits, standard binary operators (+, ×, the "^" operator that raises to a power), and parentheses. We define the length of a formula as the number of characters in the formula. Specifically, each operator, decimal digit, or parentheses counts as one character.

   (*Hint:* Think about whether it's possible to enumerate the set of possible arithmetic formulas. How would you know when to stop?)

   Solution: Yes it is possible to write such a program.

   We already know one way to write a formula for n, which is to just write the number n (with no operators). Let the length of this formula in characters be l. In order to find the *shortest* formula we simply need to search among formulae that have length at most l.

   Since there are a finite number of formulas of length at most l, we can write a program that iterates over all of them. For example, if we treat each character as a byte or an 8-bit number, the whole formula becomes a binary integer of length at most $8l$, so we can simply iterate over all binary numbers up to 2^{8l} and for each one check if it is a valid formula.

   For each formula that we encounter we can compute its value in finite time (since there are no loop/control structures in formula). Therefore we can check whether it computes n, and then among those that do compute n we find the smallest one.

   (b) Now say you wish to write a program that, given a natural number input n, finds another program (e.g. in Java or C) which prints out n. The discovered program should have the minimum execution-time-plus-length of all the programs that print n. Execution time is measured by the number of CPU instructions executed, while "length" is the number of characters in the source code. Can this be done?

   (*Hint:* Is it possible to tell whether a program halts on a given input within t steps? What can you say about the execution-time-plus-length of the program if you know that it does not halt within t steps?)

   Solution: Yes. Again it is possible to write such a program.

   As before, given a number n, there is one program that we know can definitely write n, which is the program that prints the digits of n one by one. Let the length plus running time of this program be l. We only need to check programs that have a length of at most l and a running time of at most l, since otherwise their running time plus length would be bigger than l.

   Similar to the previous part, we can iterate over all programs of length at most l (by treating each one as a large binary integer and checking each one's validity by e.g. compiling it). For each such program, we then run it for at most l steps. If it takes more time, we stop executing it and go to the next program, otherwise in at most l steps we see its output and we can check whether it is equal to n or not.

   Now among all programs that have length at most l and execute for at most l steps and print n we find the one that has the shortest length plus execution time.

3. (a) Explain why the notion of the "smallest positive integer that cannot be defined in under 280 characters" is paradoxical.

Solution: Since there are only a finite number of characters then there are only a finite number of positive integers that can be defined in under 280 characters. Therefore there must be positive integers that are not definable in 280 characters and by the well-ordering principle there is a smallest member of that set. However the statement “the smallest positive integer not definable in under 280 characters” defines the smallest such an integer using only 67 characters (including spaces). Hence, we have a paradox (called the Berry Paradox).

(b) Prove that for any length n, there is at least one string of bits that cannot be compressed to less than n bits.

Solution: The number of strings of length n is 2^n. The number of strings shorter than length n is $\sum_{i=0}^{n-1} 2^i$. We know that sum is equal to $2^n - 1$ (remember how binary works). Therefore the cardinality of the set of strings shorter than n is smaller than the cardinality of strings of length n. Therefore there must be strings of length n that cannot be compressed to shorter strings.

(c) Say you have a program K that outputs the Kolmogorov complexity of any input string. Under the assumption that you can use such a program K as a subroutine, design another program P that takes an integer n as input, and outputs the length-n binary string with the highest Kolmogorov complexity. If there is more than one string with the highest complexity, output the one that comes first lexicographically.

Solution: We write such a program as follows:

```
def P(n):
  complex_string = "0" * n
  for j in range(1, 2 ** n):
    # some fancy Python to convert j into binary
    bit_string = "0:b".format(j)
    # length should now be n characters
    bit_string = (n - len(bit_string)) * "0" + bit_string
    if K(bit_string) > K(complex_string):
      complex_string = bit_string
   return complex_string
```

(d) Let’s say you compile the program P you just wrote and get an m bit executable, for some $m \in \mathbb{N}$ (i.e. the program P can be represented in m bits). Prove that the program P (and consequently the program K) cannot exist.

(*Hint*: Consider what happens when P is given a very large input n.)

Solution: We know that for every value of n there must be an incompressible string. Such an incompressible string would have a Kolmogorov complexity greater than or equal to its actual length. Therefore our program P must return an incompressible string. However, suppose we choose size n_k such that $n_k \gg m$. Our program $P(n_k)$ will output a string x of length n_k that is not compressible meaning $K(x) \geq n_k$. However we have designed a program that outputs x using fewer bits than n_k. This is a contradiction. Therefore K cannot exist.