CS 70 Discrete Mathematics and Probability Theory
Summer 2018 Sinho Chewi and Vrettos Moulos

DIS 4A

1 Hello World!

Determine the computability of the following tasks. If it's not computable, write a reduction or self-reference proof. If it is, write the program.

(a) You want to determine whether a program P on input x prints "Hello World!". Is there a computer program that can perform this task? Justify your answer.

(b) You want to determine whether a program P prints "Hello World!" before running the kth line in the program. Is there a computer program that can perform this task? Justify your answer.

(c) You want to determine whether a program P prints "Hello World!" in the first k steps of its execution. Is there a computer program that can perform this task? Justify your answer.

Solution:

(a) Uncomputable. We will reduce `TestHalt` to `PrintsHW`(P,x).

```
TestHalt(P, x):
  P'(x):
    run P(x) while suppressing print statements
    print("Hello World!")
  if PrintsHW(P', x):
    return true
  else:
    return false
```

If `PrintsHW` exists, `TestHalt` must also exist by this reduction. Since `TestHalt` cannot exist, `PrintsHW` cannot exist.

(b) Uncomputable. Reduce `PrintsHW`(P,x) from part (a) to this program `PrintsHWByK`(P,x,k).

```
PrintsHW(P, x):
  # Find all "return" statements in P and put these
  # line numbers in set return_statements
  return_statements = []
  for i in range(len(P)):
    if isReturnStatement(i):
      return_statements.append(i)
```

```
# For each "return" statement, check if "Hello World!"
# is printed
for r in return_statements:
  if PrintsHWByK(P, x, r):
    return true
return false
```

(c) Computable. You can simply run the program until k steps are executed. If P has halted by then, return true. Else, return false.

The reason that part (b) is uncomputable while part (c) is computable is that it's not possible to determine if we ever execute a specific line because this depends on the logic of the program, but the number of computer instructions can be counted.

2 Code Reachability

Consider triplets (M, x, L) where

```
M is a Java program
x is some input
L is an integer
```

and the question of: if we execute $M(x)$, do we ever hit line L?

Prove this problem is undecidable.

Solution:

Suppose we had a procedure that could decide the above. Consider the following program for deciding whether $M(x)$ halts:

```
main():
run M(x);
print('hello')
```

Then we ask, is `print('hello')` ever executed?

If so, it means that $M(x)$ halts. Otherwise, $M(x)$ infinite looped.

3 Kolmogorov Complexity

Compression of a bit string x of length n involves creating a program shorter than n bits that returns x. The Kolmogorov complexity of a string $K(x)$ is the length of shortest program that returns x (i.e. the length of a maximally compressed version of x).

(a) Explain why "the smallest positive integer not definable in under 100 characters" is paradoxical.

(b) Prove that for any length n, there must be at least one bit string that cannot be compressed.

(c) Imagine you had the program K, which outputs the Kolmogorov complexity of string. Design a program P that when given integer n outputs the bit string of length n with the highest Kolmogorov complexity. If there are multiple strings with the highest complexity, output the lexicographically first (i.e. the one that would come first in a dictionary).

(d) Suppose the program P you just wrote can be written in m bits. Show that P and by extension, K, cannot exist, for a sufficiently large input n.

(e) Consider the program I, which can be written in m bits, that when given any input string x returns true if x is incompressible and returns false otherwise. Show that such a program cannot exist.

Solution:

(a) 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 100 characters. Therefore there must be positive integers that are not definable in 100 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 100 characters" defines the smallest such an integer using only 67 characters (including spaces). Hence, we have a paradox (called the Berry Paradox).

(b) 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) 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) 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.

(e) We prove K does not exist by writing a program S that will output a string x where $|S| < K(x)$. This would be a contradiction and so we must conclude K does not exist.

```
def S():
  i = 1
  while i > 0:
    for j in range(1, 2 ** i):
      bit_string = "0:b".format(j)
      bit_string = (i - len(bit_string)) * "0" + bit_string
      if I(bit_string) and len(bit_string) > m + c
        return bit_string
    i += 1
```

The program S must take a finite number of bits. We select a constant c such that it is larger than the length of S. Therefore the total program including S and K must take less than $m+c$ bits. S goes through every binary strings and returns the first incompressible one that is longer than $m+c$ bits. Notice the contradiction? We have found a string x that I says cannot be compressed. However, we have a program of length less than $|x|$ that outputs x, satisfying our definition of compression. Therefore I cannot exist.