CS3052 Practical 1

170011474

March 2020

1 Q1

1.1 Prove YESONSTRING $\leq_T$ HALTSONSTRING

```
from universal import universal
def alterYesToHalt(inString):
    (progString, newInString) = utils.DESS(inString)
    val = universal(progString, newInString)
    if val == 'yes':
        return 'halted'
    else:
        utils.loop()

from haltsOnString import haltsOnString
def yesViaHalts(progString, inString):
    singleStr = utils.ESS(progString, inString)
    return haltsOnString(rf('alterYesToHalt.py'), singleStr)
```

Let P and I be arbitrary inputs to `yesViaHalts`. Constructs P' and I' respectively as altered versions of P and I. Computes v = P'(I') and then returns arguments in correct form so they can be consumed by the rest of `haltsOnString`. Therefore, a reduction is complete.

1.2 Prove HALTSONSTRING $\leq_T$ HALTSONSOME by Ignoring Input

```
from universal import universal
def ignoreInput(inString):
    progString = rf('progString.txt')
    newInString = rf('inString.txt')
    return universal(progString, newInString)

from haltsOnSome import haltsOnSome
def haltsViaSome(progString, inString):
    utils.writeFile('progString.txt', progString)
    utils.writeFile('inString.txt', inString)
    return haltsOnSome(rf('ignoreInput.py'))
```

We are given an oracle that solves `HALTSVIASOME`, and now we wish to know whether P'(I') = "yes" for some particular P' and I'. As suggested above, P' and I' are stored in `progString.txt` and `inString.txt`. The returned value from `ignoreInput` is always P'(I'). Therefore, `HALTSVIASOME` can be reduced to `HALTSONSOME`.

1.3 Prove HALTSONSOME $\leq_T D_1$

```
from computesD1 import computesD1
from alterYesToInString import alterYesToInString
from generateAllStrings import generateAllStrings

def alterYesToInString(inString):
    progString, newInString = utils.DESS(inString)
    if universal(progString, newInString) == "yes":
        return inString
    else:
        return null

def haltsOnSomeViaD1(progString):
    str_list = generateAllStrings()
    for line in str_list:
        inString = utils.ESS(progString, line)
        val = computesD1("alterYesToInString.py", inString)
        if val == "yes":
            return "yes"
    return "no"
```

`generateAllStrings()` generates a list of all possible strings from ASCII alphabet. Each one is considered and is passed to `computesD1`. Let P be an arbitrary input to `HaltsOnSome`. Constructs P', an altered version of P, that works as follows: computes v = P'(I'). If v == "yes", then returns the `inString` to be consumed by `computesD1`. Therefore, `computesD1` returns "yes" if and only if `HaltsOnSome` = "yes".

2 Q2

Formal Definition of Rice's Theorem: If P is a non-trivial property, and the language holding the property L_p, is recognized by turing machine M, then $L_p = \{< M > | L(M) \in P\}$ is undecidable.

Description and Properties Property of languages, P, is simply a set of languages. If any language belongs to P ($L \in P$), it is said that L satisfies the property P.

A property is called to be trivial if either it is not satisfied by any recursively enumerable languages, or if it is satisfied by all recursively enumerable languages.

A non-trivial property is satisfied by some recursively enumerable languages and are not satisfied by others. Formally speaking, in a non-trivial property, where $L \in P$, both the following properties hold:

1. **Property 1** - There exists Turing Machines, M1 and M2 that recognize the same language, i.e. either ($< M1 >, < M2 > \in L$) or ($< M1 >, < M2 > \notin L$)
2. **Property 2** - There exists Turing Machines M1 and M2, where M1 recognizes the language while M2 does not, i.e. $< M1 > \in L$ and $< M2 > \notin L$

2.1 Prove D_{2a} is undecidable

Let S be the set of functions that map ASCII strings to ASCII strings and return "yes" if it accepts a finite number of inputs and "no" otherwise. Since S includes at least one computable function($IsEvenLessThan10$, which only accepts even positive numbers less than 10, so it only accepts 2, 4, 6, 8), and there is also one which does not belong to this property (a function that accepts everything, so accepts infinite inputs). So it is non-trivial. Hence the conditions of Rice's theorem are fulfilled and we conclude that the D_{2a} is undecidable.

2.2 Prove D_{2b} is undecidable

This one is simple proved if we use the same logic as D_{2a}.

Let S be the set of functions that map ASCII strings to ASCII strings and return "yes" if it accepts an infinite number of inputs and "no" otherwise. Since S includes at least one computable function(a function that accepts everything, so accepts infinite inputs), and there is also one which does not belong to this property ($IsEvenLessThan10$, which only accepts even positive numbers less than 10, so it only accepts 2, 4, 6, 8). So it is non-trivial. Hence the conditions of Rice's theorem are fulfilled and we conclude that the D_{2b} is undecidable.

2.3 Prove D_{2c} is undecidable

Since there is at least one computable function that can return "yes" if it accepts representations of prime numbers and "no" otherwise, but there is also at least one computable function that rejects every representation of prime number, or there is a function that rejects everything. So the problem is non-trivial. Hence the conditions of Rice's theorem are fulfilled and we conclude that the D_{2c} is undecidable.

3 Q3

3.1 Prove HALTSONSTRING $\leq_T$ D_3 By Ignoring Input

```
from universal import universal
def alter(inString):
    progString = rf('progString.txt')
    newInString = rf('inString.txt')

    val = universal(progString, newInString)
    if val == "yes":
        return 1
    else:
        return -1

def HaltsViaD3(progString, inString):
    utils.writeFile('progString.txt', progString)
    utils.writeFile('inString.txt', inString)
    return computesD3("alter.py")
```

Let P be an arbitrary input to `HaltsViaD3`. Constructs a P' as an altered version of P. For ignoring inputs, `progString` and `inString` are written into files. P' is passed to `computesD3`. Do

the following work: computes v = P'(I') where I' is given by `computesD3`. Reads inputs from files and returns natural number consumed by the rest of `computesD3` if and only if halts, otherwise return a non-natural number. Therefore, a reduction from `HaltsOnSome` to `computesD3` is finished.