**CS350 – Reductions**

**Purpose:** Computer scientists use reductions (using one problem to solve another problem) to show that a problem is hard. The purpose of this module is for you to use reductions to show that problems are uncomputable.

**Knowledge:** This module will help you become familiar with the following content knowledge:

- How to determine what information a reduction gives us.
- How to use reductions to show problems are uncomputable.

**Activity:** With your group perform the following tasks and answer the questions. You will be reporting your answers back to the class in 90 minutes.

1. We will start with a review of some important concepts.

```
from universal import universal
def alterGAGAtoTATA(inString):
    (progString, newInString) = utils.DESS(inString)
    val = universal(progString, newInString)
    if val == 'GAGA':
        return 'TATA'
    else:
        return val
```

Give the output of each of the following:

```
>>>alterGAGAtoTATA(utils.ESS('def f(S): return S', 'CAGT'))

>>> alterGAGAtoTATA(utils.ESS('def f(S): return S', 'GAGA'))

>>> alterGAGAtoTATA(utils.ESS('def f(S): return S+S', 'GA'))

>>> alterGAGAtoTATA(utils.ESS('def f(S): return S+S', 'GAGA'))
```

2. Given the code for `ignoreInput.py`, give the output for the following sequence of commands.

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

```
>>> prog = rf('longerThan1K.py')
>>> utils.writeFile('progString.txt', prog)
>>> utils.writeFile('inString.txt', 'abcdefghij')
>>> ignoreInput(5000*'x') # param is string of 5000 x's
```

3. A *reduction* is where we use the solution of one problem to solve another problem.

   Here is the formal definition:
   Let $F$ and $G$ be computational problems. We say $F$ has a Turing reduction to $G$ if we can write a Python program solving $F$, after first assuming the existence of a program solving $G$.
   Notation is: $F \leq_T G$

   How could we use a program EndsInZero, which returns true or false on whether a string represents a number ending with a zero, to solve the problem MultipleOf10, which determines if a string represents a number which is a multiple of ten? This would reduce MultiplesOf10 to EndsInZero.

4. The problem YesOnString, which determines whether a program returns "yes" when run on the given string, reduces to GAGAOnString, which determines whether a program returns "GAGA" when run on the given string.

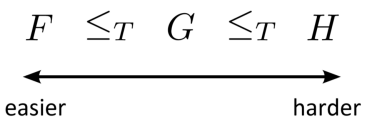
   Consider the following programs:

```
def alterYesToGAGA(inString):
    (progString, newInString) = utils.DESS(inString)
    val = universal(progString, newInString)
    if val == 'yes':
        return 'GAGA'
    else:
        return 'no'
```

```
from GAGAOnString import GAGAOnString # oracle function
def yesViaGAGA(progString, inString):
    singleString = utils.ESS(progString, inString)
    return GAGAOnString(rf('alterYesToGAGA.py'), singleString)
```

   Explain why this shows that GAGAOnString is uncomputable (just like YesOnString).

5. Think of Turing reductions as ordering problems according to "hardness".




   If we know that $G$ is uncomputable does that tell us anything about the computability of either $F$ or $H$?
   If we know that $G$ is computable does that tell us anything about the computability of either $F$ or $H$?

6. The "ignore input" trick can be used for many reductions. For example, consider the reduction from YesOnString to YesOnEmpty:

```
from yesOnEmpty import yesOnEmpty # oracle function
def yesViaEmpty(progString, inString):
   utils.writeFile('progString.txt', progString)
   utils.writeFile('inString.txt', inString)
   return yesOnEmpty(rf('ignoreInput.py'))
```

   Explain why this is a correct reduction.

   How would you modify this reduction to show YesOnSome is undecidable? YesOnSome returns "yes" if $P(I)$ is defined and returns "yes" for at least one $I$ and no otherwise.

7. Here is another example:

```
from universal import universal
def alterYesToHalt(inString):
   (progString, newInString) = utils.DESS(inString)
   val = universal(progString, newInString)
   if val == 'yes':
      # return value is irrelevant, since returning any string halts
      return 'halted'
   else:
      # deliberately enter infinite loop
      utils.loop()
```

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

   Explain why this shows that HaltOnString is undecidable.

8. An important family of problems is $Computes_F$ in which a problem determines if a program computes a given function $F$.

   Consider the following two problems:

| **Problem** IsEven |
| --- |
| • **Input:** An integer $M$, as an ASCII string in decimal notation.<br>• **Solution:** "yes" if $M$ is even, and "no" otherwise. |

| **Problem** ComputesIsEven |
| --- |
| • **Input:** A program $P$, as an ASCII string.<br>• **Solution:** "yes" if $P$ computes IsEven, and "no" otherwise. |

   Is the problem IsEven decidable?
   Do you think ComputesIsEven decidable? To help you answer that consider the following programs. Can you determine which of these are instances of the ComputesIsEven problem? You may find this more challenging than you think.

```
def P1(inString):
    n = int(inString)
    if n%2==0: return 'yes'
    else: return 'no'

def P2(inString):
    if inString[-1] in '02468': return 'yes'
    else: return 'no'
```

```
def P3(inString):
    n = int(inString)
    if (2*n+1)%4==1: return 'yes'
    else: return 'no'

def P4(inString):
    n = int(inString)
    if (3*n+1)%5==1: return 'yes'
    else: return 'no'
```

9. It is time to show that $Computes_F$ is undecidable for any computable problem $F$:

```
from universal import universal
from F import F
from G import G # G is any computable function different to F
def alterYesToComputesF(inString):
   progString = rf('progString.txt')
   newInString = rf('inString.txt')
   val = universal(progString, newInString)
   if val == 'yes':
      return F(inString)
   else:
      return G(inString)
```

```
from computesF import computesF # oracle function
def yesViaComputesF(progString, inString):
   utils.writeFile('progString.txt', progString)
   utils.writeFile('inString.txt', inString)
   val = computesF(rf('alterYesToComputesF.py'))
   if val == 'yes':
      return 'yes'
   else:
      return 'no'
```

   Explain why this shows that $Computes_F$ is undecidable.

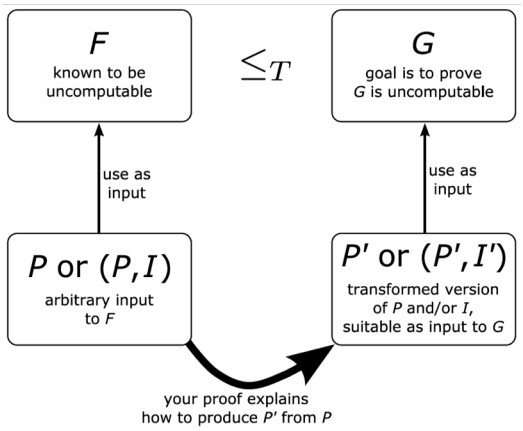
10. Consider the following problem:

**Problem** HALTSBEFORE100

- **Input:** A Turing machine $M$.
- **Solution:** "yes" if there exists some input $I$ for which $M$ halts before completing 100 steps; and "no" otherwise.

   Is HaltsBefore100 decidable or undecidable? Why?

11. The text shows two techniques to perform reductions. The recipe for the first technique is illustrated as follows:




Show that ContainsZOnAll(P), which determines whether all inputs of P produce an output which contains the character "Z", is uncomputable. To do this we can show that YesOnAll $\leq_T$ ContainsZOnAll. Given an arbitrary input $P$ for YesOnAll, how would you construct $P'$ so that ContainsZOnAll(P') = "yes" if an only if YesOnAll(P)="yes"?

12. Another technique that you may find easier to understand is write explicit python programs. You write an "alteration" program which transforms the output of F to the output of G. Additionally, you write a "reduction" program that solves F by using G's program on the alteration program.

    Here is an example of that shows that NumCharsOnString is uncomputatable. NumCharsOnString takes input P and I returns the number of characters in the output of P(I) if P(I) is defined and returns "no" otherwise. The "alteration" and "reduction" below shows that YesOnString $\leq_T$ NumCharsOnString:

```
from universal import universal
def alterYesToNumChars(inString):
   (progString, newInString) = utils.DESS(inString)
   val = universal(progString, newInString)
   if val == 'yes':
      # return a string with three characters
      return 'xxx'
   else:
      # return a string with two characters
      return 'xx'
```

```
from numCharsOnString import numCharsOnString # oracle function
def yesViaNumChars(progString, inString):
   singleString = utils.ESS(progString, inString)
   val = numCharsOnString(rf('alterYesToNumChars.py'), \
                          singleString)
   if val == '3':
      return 'yes'
   else:
      return 'no'
```

    Write programs alterYesToZ and yesViaContainsZ which will prove that containsZOnAll is undecidable.

Complete the following assignments for grading. Each should be done individually but you may consult with a classmate to discuss strategies.

**Assignment 1**:
Complete exercise 7.4 on p140 in the text.

**Criteria for Success:** For each of the three cases, you need to explain what you can deduce about problems F,G, and D. Also clearly explain if you can not deduce anything about any of the problems.

**Assignment 2**:
Complete exercise 7.5 on p140 in the text.

**Criteria for Success:** For each of the five problems, you have a complete clear proof that the problem is undecidable. You do this by reducing a problem we already know is undecidable to the given problem. I suggest you use the "technique 2" of writing two Python functions to perform the reduction. I reduced either `yesOnString` or `yesOnEmpty` to the given problems to show they are undecidable.

Your Python functions could look something like:

```
def alterP(inString):  #alter the program P that is input to yesOnString
   # execute P(I)
   val = universal(rf('progString.txt'),rf('inString.txt'))
   if val = "yes":
       return ____________
   else:
       return ____________

def yesOnStringViaNewProblem(P,I):
   utils.writeFile('progString.txt',P)
   utils.writeFile('inString.txt',I)
   # use the new decision problem on your alterP program
   # to answer yesOnString problem
```

**Assignment 3**:
Complete exercise 7.7 on p141 in the text.

**Criteria for Success:** You have a clear explanation why this reduction is not correct.

**Assignment 4**:
Complete exercise 7.15 on p142 in the text.

**Criteria for Success:** You have a complete clear proof that the problem is uncomputable. EFC is not a decision problem but the solution should be similar to your previous reductions nonetheless.

Submit your written answers in Canvas for grading.