Simon

May 29, 2020

[16]:
```
import numpy as np
import inspect

def f_0(a, b): #s = 10
    no_args = 2
    string_ = str(a)+str(b)
    dict_={}
    dict_['00'] = dict_['10'] = format(1, '#0'+str(no_args+2)+'b')
    dict_['01'] = dict_['11'] = format(2, '#0'+str(no_args+2)+'b')
    return dict_[string_]

def f_1(a, b): #s = 01
    no_args = 2
    string_ = str(a)+str(b)
    dict_={}
    dict_['00'] = dict_['01'] = format(1, '#0'+str(no_args+2)+'b')
    dict_['10'] = dict_['11'] = format(2, '#0'+str(no_args+2)+'b')
    return dict_[string_]

def f_2(a, b): #s = 11
    no_args = 2
    string_ = str(a)+str(b)
    dict_={}
    dict_['00'] = dict_['11'] = format(1, '#0'+str(no_args+2)+'b')
    dict_['01'] = dict_['10'] = format(2, '#0'+str(no_args+2)+'b')
    return dict_[string_]

def f_3(a, b, c): #s = 110
    no_args = 3
    string_ = str(a)+str(b)+str(c)
    dict_={}
    dict_['000'] = dict_['110'] = format(1, '#0'+str(no_args+2)+'b')
    dict_['001'] = dict_['111'] = format(2, '#0'+str(no_args+2)+'b')
    dict_['010'] = dict_['100'] = format(3, '#0'+str(no_args+2)+'b')
    dict_['011'] = dict_['101'] = format(4, '#0'+str(no_args+2)+'b')
    return dict_[string_]
```

```
def f_4(a, b, c): #s = 010
    no_args = 3
    string_ = str(a)+str(b)+str(c)
    dict_={}
    dict_['000'] = dict_['010'] = format(1, '#0'+str(no_args+2)+'b')
    dict_['001'] = dict_['011'] = format(2, '#0'+str(no_args+2)+'b')
    dict_['100'] = dict_['110'] = format(3, '#0'+str(no_args+2)+'b')
    dict_['101'] = dict_['111'] = format(4, '#0'+str(no_args+2)+'b')
    return dict_[string_]

def f_5(a, b, c): #s = 111
    no_args = 3
    string_ = str(a)+str(b)+str(c)
    dict_={}
    dict_['000'] = dict_['111'] = format(1, '#0'+str(no_args+2)+'b')
    dict_['001'] = dict_['110'] = format(2, '#0'+str(no_args+2)+'b')
    dict_['010'] = dict_['101'] = format(3, '#0'+str(no_args+2)+'b')
    dict_['011'] = dict_['100'] = format(4, '#0'+str(no_args+2)+'b')
    return dict_[string_]

def f_6(a, b, c, d): #s = 1010
    no_args = 4
    string_ = str(a)+str(b)+str(c)+str(d)
    dict_={}
    dict_['0000'] = dict_['1010'] = format(1, '#0'+str(no_args+2)+'b')
    dict_['0001'] = dict_['1011'] = format(2, '#0'+str(no_args+2)+'b')
    dict_['0010'] = dict_['1000'] = format(3, '#0'+str(no_args+2)+'b')
    dict_['0011'] = dict_['1001'] = format(4, '#0'+str(no_args+2)+'b')
    dict_['0100'] = dict_['1110'] = format(5, '#0'+str(no_args+2)+'b')
    dict_['0101'] = dict_['1111'] = format(6, '#0'+str(no_args+2)+'b')
    dict_['0110'] = dict_['1100'] = format(7, '#0'+str(no_args+2)+'b')
    dict_['0111'] = dict_['1101'] = format(8, '#0'+str(no_args+2)+'b')
    return dict_[string_]

def f_7(a, b, c, d): #s = 1011
    no_args = 4
    string_ = str(a)+str(b)+str(c)+str(d)
    dict_={}
    dict_['0000'] = dict_['1011'] = format(1, '#0'+str(no_args+2)+'b')
    dict_['0001'] = dict_['1010'] = format(2, '#0'+str(no_args+2)+'b')
    dict_['0010'] = dict_['1001'] = format(3, '#0'+str(no_args+2)+'b')
    dict_['0011'] = dict_['1000'] = format(4, '#0'+str(no_args+2)+'b')
    dict_['0100'] = dict_['1111'] = format(5, '#0'+str(no_args+2)+'b')
    dict_['0101'] = dict_['1110'] = format(6, '#0'+str(no_args+2)+'b')
    dict_['0110'] = dict_['1101'] = format(7, '#0'+str(no_args+2)+'b')
    dict_['0111'] = dict_['1100'] = format(8, '#0'+str(no_args+2)+'b')
    return dict_[string_]
```

```
def f_8(a, b, c, d): #s = 1111
    no_args = 4
    string_ = str(a)+str(b)+str(c)+str(d)
    dict_={}
    dict_['0000'] = dict_['1110'] = format(1, '#0'+str(no_args+2)+'b')
    dict_['0001'] = dict_['1111'] = format(2, '#0'+str(no_args+2)+'b')
    dict_['0010'] = dict_['1100'] = format(3, '#0'+str(no_args+2)+'b')
    dict_['0011'] = dict_['1101'] = format(4, '#0'+str(no_args+2)+'b')
    dict_['0100'] = dict_['1010'] = format(5, '#0'+str(no_args+2)+'b')
    dict_['0101'] = dict_['1011'] = format(6, '#0'+str(no_args+2)+'b')
    dict_['0110'] = dict_['1000'] = format(7, '#0'+str(no_args+2)+'b')
    dict_['0111'] = dict_['1001'] = format(8, '#0'+str(no_args+2)+'b')
    return dict_[string_]

def f_9(a, b, c, d, e): #s = 11111
    no_args = 5
    string_ = str(a)+str(b)+str(c)+str(d)+str(e)
    dict_={}
    dict_['00000'] = dict_['11111'] = format(1, '#0'+str(no_args+2)+'b')
    dict_['00001'] = dict_['11110'] = format(2, '#0'+str(no_args+2)+'b')
    dict_['00010'] = dict_['11101'] = format(3, '#0'+str(no_args+2)+'b')
    dict_['00011'] = dict_['11100'] = format(4, '#0'+str(no_args+2)+'b')
    dict_['00100'] = dict_['11011'] = format(5, '#0'+str(no_args+2)+'b')
    dict_['00101'] = dict_['11010'] = format(6, '#0'+str(no_args+2)+'b')
    dict_['00110'] = dict_['11001'] = format(7, '#0'+str(no_args+2)+'b')
    dict_['00111'] = dict_['11000'] = format(8, '#0'+str(no_args+2)+'b')
    dict_['01000'] = dict_['10111'] = format(9, '#0'+str(no_args+2)+'b')
    dict_['01001'] = dict_['10110'] = format(10, '#0'+str(no_args+2)+'b')
    dict_['01010'] = dict_['10101'] = format(11, '#0'+str(no_args+2)+'b')
    dict_['01011'] = dict_['10100'] = format(12, '#0'+str(no_args+2)+'b')
    dict_['01100'] = dict_['10011'] = format(13, '#0'+str(no_args+2)+'b')
    dict_['01101'] = dict_['10010'] = format(14, '#0'+str(no_args+2)+'b')
    dict_['01110'] = dict_['10001'] = format(15, '#0'+str(no_args+2)+'b')
    dict_['01111'] = dict_['10000'] = format(16, '#0'+str(no_args+2)+'b')
    return dict_[string_]
```

[17]:
```
def getDecimalNo(arr_1, arr_2):
    integer = 0
    n = 2*len(arr_1)
    final_arr = np.concatenate((arr_1,arr_2))
    for i in range(0,n):
        integer = integer + (2**i)*final_arr[n-1-i]
    return integer
```

[18]:
```
def create_Uf(f, n):
```

```
    binary1 = format(1, '#0'+str(n+2)+'b') #results in a binary string␣
↪representing 1 with adequate number of 0s given by arity of f
    binary2 = format(0, '#0'+str(n+2)+'b') #results in a binary string␣
↪representing 0 with adequate number of 0s given by arity of f
    result = {}
    query_list = [int(d) for d in str(binary2)[2:]] #bitstring to send function
    for i in range(pow(2,n)):
        result[binary2] = f(*query_list)
        integer_sum = int(binary1, 2) + int(binary2, 2)
        binary2 = format(integer_sum, '#0'+str(n+2)+'b') #updating the second␣
↪binary string by adding 1 to get the next input sequence to send f
        query_list = [int(d) for d in str(binary2)[2:]]
    #with the above code we obtained the results of f for all 2**N bitstrings,␣
↪now we proceed to get Uf for 2**2N bitstrings

    U_f = np.zeros((2**(2*n), 2**(2*n)))
    binary1 = format(1, '#0'+str(n+2)+'b') #results in a binary string␣
↪representing 1 with adequate number of 0s given by arity of f
    binary2 = format(0, '#0'+str(n+2)+'b') #results in a binary string␣
↪representing 0 with adequate number of 0s given by arity of f
    for i in range(pow(2,n)):
        b = format(0, '#0'+str(n+2)+'b')
        bin_num1_arr = np.array([int(d) for d in str(result[binary2])[2:]])␣
↪#f(x)

        for i in range(pow(2,n)):
            bin_num2_arr = np.array([int(d) for d in str(b)[2:]]) #b
            f_x_b = np.add(bin_num1_arr,bin_num2_arr) %2

            column = getDecimalNo(np.array([int(d) for d in str(binary2)[2:]]),␣
↪f_x_b)
            row = getDecimalNo(np.array([int(d) for d in str(binary2)[2:]]), np.
↪array([int(d) for d in str(b)[2:]]))

            U_f[row][column] = 1

            integer_sum = int(binary1, 2) + int(b, 2)
            b = format(integer_sum, '#0'+str(n+2)+'b')
        integer_sum = int(binary1, 2) + int(binary2, 2)
        binary2 = format(integer_sum, '#0'+str(n+2)+'b')
    return U_f
```

[19]:
```
from qiskit import QuantumCircuit, execute, Aer
from qiskit.quantum_info.operators import Operator

def run_program(f, n):
```

```
    U_f = create_Uf(f, n)
    N = n*2

    qubit_list=[]

    for i in range(0,int(N)):
        qubit_list.append(i)

    qc = QuantumCircuit(N,N) #create qc with the number of qubits required

    #adding Hadamard gates to all main qubits
    qc.h(range(n,N))

    #create the UF gate
    uf_gate = Operator(U_f)
    qc.unitary(uf_gate, qubit_list, label = 'uf')

    #adding the remaining Hadamards to the main qubits
    qc.h(range(n,N))

    qc.measure(qubit_list,qubit_list)

    #print(qc.draw())

    simulator = Aer.get_backend("qasm_simulator")
    job = execute(qc, simulator, shots=10000)

    try:
        result = job.result()
        counts=result.get_counts(qc)
    except TimeoutError:
        print("Could not obtain results for N: {}".format(N/2))
        return[]

    return counts
```

[20]:
```
import time

#had difficulty coding this section on solving the linear equations,
#took help from a friend who did the course the previous quarter to understand␣
 ↪the functions and implement them
#also referenced book on solving linear equations with mod 2 using gaussian␣
 ↪elimination and back substitution

def isAllZeros(list_):
    for i in list_:
        if i!=0:
```

```
            return False
    return True

def addRow(vectors, z, MSBDict):
    while isAllZeros(z) is False:
        msbOne = z.index(1)
        if msbOne in MSBDict.keys():
            otherRow = MSBDict[msbOne]
            z = list(np.add(np.array(vectors[otherRow]),np.array(z))%2) #update␣
 ↪z (adding those two rows with same msb to get reduced row echolon form)
        else:
            break

    if(isAllZeros(z)):
        return vectors, MSBDict
    else:
        msbOne = z.index(1)
        keysList = list(MSBDict.keys())
        row = 0
        for row in range(len(keysList)):
            if keysList[row] > msbOne:
                break
            row+=1
        vectors.insert(row,z)
        MSBDict[z.index(1)]=row

        return vectors, MSBDict

def addLastVector(vectors):
    lastRow = [0 for i in range(len(vectors[0]))]
    lastOne = 0
    for i in range(len(vectors)):
        if vectors[i][i]==1:
            lastOne = i+1
        else:
            break
    lastRow[lastOne]=1 #insert 1 at appropriate msb position
    vectors.insert(lastOne, lastRow) #insert the lastRow in the proper index␣
 ↪corresponding to lastOne+1
    return lastRow, vectors

def getS(f):
    MSBDict = {}
    vectors = []
    n = len(inspect.signature(f).parameters)

    start = time.time()
```

```
    results_dict = run_program(f, n)
    for key_ in results_dict:
        results=[]
        for val in key_[0:n]:
            results.append(int(val))
        if(len(vectors) != n-1):
            vectors, MSBDict = addRow(vectors, results, MSBDict)
        else:
            break

    end = time.time()
    total_time = end - start

    newRow, vectors = addLastVector(vectors)
    arr_1 = np.array(vectors)
    arr_2 = np.zeros(n)
    arr_2[newRow.index(1)] = 1
    y = np.linalg.solve(arr_1,arr_2)
    y = [i%2 for i in y]
    s = ''
    for i in range(len(y)):
        s+=str(int(y[i]))
    return s, total_time
```

[25]:
```
print(getS(f_0))
print(getS(f_1))
print(getS(f_2))
print(getS(f_3))
print(getS(f_4))
print(getS(f_5))
print(getS(f_6))
print(getS(f_7))
print(getS(f_8))
```

```
('10', 0.03384089469909668)
('01', 0.027615070343017578)
('11', 0.024251699447631836)
('110', 0.04090285301208496)
('010', 0.037837982177734375)
('111', 0.038137197494506836)
('1010', 0.11496496200561523)
('1011', 0.11623072624206543)
('1110', 0.11368703842163086)
```

[26]:
```
print(getS(f_0))
print(getS(f_1))
```

```
print(getS(f_2))
print(getS(f_3))
print(getS(f_4))
print(getS(f_5))
print(getS(f_6))
print(getS(f_7))
print(getS(f_8))
```

```
('10', 0.03763413429260254)
('01', 0.02613997459411621)
('11', 0.02495598793029785)
('110', 0.03937673568725586)
('010', 0.038710832595825195)
('111', 0.0372920036315918)
('1010', 0.11119985580444336)
('1011', 0.10656905174255371)
('1110', 0.11297607421875)
```

[27]:
```
print(getS(f_0))
print(getS(f_1))
print(getS(f_2))
print(getS(f_3))
print(getS(f_4))
print(getS(f_5))
print(getS(f_6))
print(getS(f_7))
print(getS(f_8))
```

```
('10', 0.03498387336730957)
('01', 0.02777385711669922)
('11', 0.02542901039123535)
('110', 0.03924918174743652)
('010', 0.04091978073120117)
('111', 0.03907203674316406)
('1010', 0.1063680648803711)
('1011', 0.11171817779541016)
('1110', 0.10549020767211914)
```

[ ]:
```

```