Домашняя работа по дисциплине Теоретические модели вычслений №2

Чуворкин Михаил А-13а-19

31 мая 2022 г.

1 Машины Тьюринга

Работу требуется выполнять в системе `turingmachine.io`.

Для сдачи заданий 1-2 требуется прикрепить файлы YAML с исходным кодом проекта. Каждый файлы должен иметь наименование задание_пункт.yml, к примеру 1_1.yml для первой задачи первого задания.

1.1 Операции с числами

Реализуйте машины Тьюринга, которые позволяют выполнять следующие операции:

1. Сложение двух унарных чисел (1 балла)
   Алгоритм:
   - Движемся вправо, пока не встретили '+'.
   - Заменяем '+' на 1 и движемся к хвосту.
   - Находим крайнюю единицу и удаляем её.
   - Движемся к голове.

```yaml
input: '||+|||'
blank: ' '


start state: q0
table:
  q0:
    '|': {R: q0}
    '+': {write: '|', R: q1}

  q1:
    '|': {R: q1}
    ' ': {L: q2}

  q2:
    '|': {write: ' ', L: ret}

  ret:
    '|': L
    ' ': {R: done}

  done:
```

2. Умножение унарных чисел (1 балл)
   Алгоритм:
   - Берем первую '|', идем в конец второго числа, ставим там '*'
   - Копируем второе число после '*' (или после последней '|', если это не первый ход цикла)

- Идем в начало первого числа, забираем еще одну '|', повторяем цикл
- Если слева от '*' стоит пустой символ, удаляем звездочку, умножение закончено

```
input: '*' # try '*', '|*|||', '||||*|||'
blank: ' '


start state: q1
table:
  # start - picks first |
  q1:
    '|': {write: ' ', R: q2}
    '*': {write: ' ', R: q11}

  # skips all | until *
  q2:
    '|': {R: q2}
    '*': {R: q3}

  # skips all until empty and writes * at the end if on the first pass
  q3:
    '|': {R: q3}
    '*': {L: q4}
    ' ': {write: '*', L: q4}

  # skip until * (in second number)
  q4:
    '|': {L: q4}
    '*': {R: q5}

  # picks | to copy, if meets * then all | are copied -- exiting loop
  q5:
    '|': {write: ' ', R: q6}
    '*': {L: q9}

  # skip all | until * (moving right)
  q6:
    '|': {R: q6}
    '*': {R: q7}

  # write | on a first empty cell
  q7:
    '|': {R: q7}
    ' ': {write: '|', L: q8}

  # skips until empty cell (prev picked) and writes | back
  q8:
    '|': {L: q8}
    '*': {L: q8}
    ' ': {write: '|', R: q5}

  # skip | in second number (moving left)
  q9:
    '|': {L: q9}
    '*': {L: q10}

  # skip | in first number (moving left)
  # if meets empty cell, repeats loop
  q10:
    '|': {L: q10}
    ' ': {R: q1}

  # erases everything except | moving right and stops
```

```
  q11:
    '|': {write: ' ', R: q11}
    '*': {write: ' ', R: done}
    ' ': {L: done} # ?? why it cannot just stay

  done:
```

1.2 Операции с языками и символами

Реализуйте машины Тьюринга, которые позволяют выполнять следующие операции:

1. Принадлежность к языку $L = \{0^n1^n2^n\}, n \geq 0$ (0.5 балла)
   Алгоритм:
   - Заменяем последний '0' на 'F', идем вправо, аналогично заменяем последнюю '1', идем вправо, заменяем последнюю '2'
   - Идем в начало слова, повторяем предыдущий пункт
   - Если после некоторого количества повторений остались все 'F', значит слово принадлежит языку - затираем все, пишем 'T'
   - Если по пути прохода встретился символ, отличный от 'F', значит слово не принадлежит языке - затираем всеб пишем 'F'

```
input: '000111222'
blank: ' '

start state: q1
table:

  q1:
    '0': {R: q1}
    '1': {L: q2}
    '2': {write: 'F', L: doneF}
    'F': {R: q1}
    ' ': {L: doneT}
  q2:
    '0': {write: 'F', R: q3}
    '1': {write: 'F', L: doneF}
    '2': {write: 'F', L: doneF}
    'F': {L: q2}
    ' ': {write: 'F', L: doneF}

  q3:
    '0': {write: 'F', L: doneF}
    '1': {R: q3}
    '2': {L: q4}
    'F': {R: q3}
    ' ': {write: 'F', L: doneF}

  q4:
    '0': {write: 'F', L: doneF}
    '1': {write: 'F', R: q5}
    '2': {write: 'F', L: doneF}
    'F': {L: q4}
    ' ': {write: 'F', L: doneF}

  q5:
    '0': {write: 'F', L: doneF}
    '1': {write: 'F', R: q5}
    '2': {R: q5}
    'F': {R: q5}
    ' ': {L: q6}

  q6:
```

```
    '0': {write: 'F', L: doneF}
    '1': {write: 'F', L: doneF}
    '2': {write: 'F', L: q7}
    'F': {L: q6}
    ' ': {write: 'F', L: doneF}

  q7:
    '0': {L: q7}
    '1': {L: q7}
    '2': {L: q7}
    'F': {L: q7}
    ' ': {R: q1}

  doneF:
    ['0', '1', '2', 'F']: {write: 'F', R: clearRF}
    ' ': {write: 'F', L: fin}

  clearRF:
    ['0', '1', '2', 'F']: {write: 'F', R: clearRF}
    ' ': {L: clearLF}
  clearLF:
    ['0', '1', '2', 'F']: {write: ' ', L: clearLF}
    ' ': {write: 'F', L: fin}

  doneT:
    'F': {write: ' ', L: doneT}
    ' ': {write: 'T', L: fin}


  fin:
```

2. Проверка соблюдения правильности скобок в строке (минимум 3 вида скобок) (0.5 балла)
Алгоритм:

   - Так как самая минимальная верная последовательность '()' - идем до первой закрывающей, ее заменяем на 'F'
   - Идем на 1 назад- там должна быть соответствующая открывающая, если она там есть - заменяем ее на 'F', иначе - ошибка
   - После замены открывающей на 'F' идем в начало последовательности и повторяем все с начала
   - Если ошибка - очищаем все, пишем 'F'
   - Если в начале стоит 'F' - очищаем последовательность из 'F'. Повторяем все с начала.
   - Если после очистки справа встречаеся пустой символ - последовательность закончилась и она верная

```
input: '({)}'
blank: ' '
start state: q1
table:

  q1:
    ['(', '{', '[']: {R}
    ')': {write: 'F', L: q2a}
    '}': {write: 'F', L: q2b}
    ']': {write: 'F', L: q2c}
    'F': {R: q1}
    ' ': {L: q5}

  q2a:
    '(': {write: 'F', L: q3}
    ['{', '[']: {write: 'F', L: clL}
    'F': {L: q2a}
```

```
  q2b:
    '{': {write: 'F', L: q3}
    ['(', '[']: {write: 'F', L: clL}
    'F': {L: q2b}

  q2c:
    '[': {write: 'F', L: q3}
    ['{', '(']: {write: 'F', L: clL}
    'F': {L: q2c}

  q3:
    ['(', ')', '{', '}', '[', ']', 'F']: {L: q3}
    ' ': {R: q4}

  tmp:
    ['(', ')', '{', '}', '[', ']', 'F', 'T', ' ']: {R: q1}

  q4:
    ['(', '{', '[']: {L: tmp}
    'F': {write: ' ', R: q4}
    ' ': {write: 'T', L: done}

  q5:
    ['(', ')', '{', '}', '[', ']', 'F']: {write: ' ', L: q5}
    ' ': {write: 'F', L: done}

  clL:
    ['(', ')', '{', '}', '[', ']', 'F']: {write: 'F', L: clL}
    ' ': {R: clR}

  clR:
    ['(', ')', '{', '}', '[', ']', 'F']: {write: ' ', R: clR}
    ' ': {write: 'F', L: done}

  done:
```

3. Поиск минимального по длине слова в строке (слова состоят из символов 1 и 0 и разделены пробелом) (1 балл)
Алгоритм:

   - Будем считать, что пробел и пустой символ - разные. В эмуляторе пробел будет обозначен нижним подчеркиванием, а пустой символ пробелом
   - Идем вправо, заменяя первые симолы на похожие (’0’ на ’o’, ’1’ на ’l’)
   - Возвращаемя в начало, повторяем замены
   - Если справа от замененного стоит пустой символ - это было самое короткое слово. Ставим справа ’*’
   - Удаляем все, что после ’*’, идем обратно, заменяем обратно символы, очищаем все, что слева от найденного слова

```
input: '10_10_11'
blank: ' '
start state: q1

table:
  q1:
    ['l','o']: {R: q1}
    '1': {write: 'l', R: q2}
    '0': {write: 'o', R: q2}
    ['_',' ']: {write: '*',R: q4}
  q2:
    ['0', '1']: {R: q2}
    '_': {R: q1}
```

```
    ' ': {L: q3}
  q3:
    ['0', '1', 'l', 'o', '_']: {L: q3}
    ' ': {R: q1}
  q4:
    ['0', '1', 'l', 'o', '_']: {R: q4}
    ' ': {L: q5}
  q5:
    ['0', '1', '_', 'l', 'o']: {write: ' ', L}
    '*': {write: ' ', L: q6}
  q6:
    'l': {write: '1', L}
    'o': {write: '0', L}
    '_': {write: ' ', L: q7}
  q7:
    ['0', '1', 'l', 'o', '_']: {write: ' ', L}
    ' ': {R: q8}
  q8:
    ' ': {R: q8}
    ['0', '1']: {L: done}
  done:
```

2 Квантовые вычисления

2.1 Генерация суперпозиций 1 (1 балл)

Дано N кубитов ($1 \leq N \leq 8$) в нулевом состоянии $|0\dots0\rangle$. Также дана некоторая последовательность битов, которое задаёт ненулевое базисное состояние размера N. Задача получить суперпозицию нулевого состояния и заданного.

$$|S\rangle = \frac{1}{\sqrt{2}}(|0\dots0\rangle + |\psi\rangle)$$

То есть требуется реализовать операцию, которая принимает на вход:

1. Массив кубитов q_s
2. Массив битов $bits$ описывающих некоторое состояние $|\psi\rangle$. Это массив имеет тот же самый размер, что и qs. Первый элемент этого массива равен 1.

Решение:
Решение будет аналогично созданию суперпозиции для векторов ϕ и ψ, только различие уже есть в первом кубите. Применяем к нему оператор Адамара. Остальные к остальным кубитам, если они равны $|1\rangle$ применяем CX, спутывая с первым.

```
circuit.h(0)
circuit.barrier()
for i in range(1, len(bits)):
    if bits[i]: circuit.cx(qr[0], qr[i])

circuit.draw(initial_state=True)
```

2.2 Различение состояний 1 (1 балл)

Дано N кубитов ($1 \leq N \leq 8$), которые могут быть в одном из двух состояний:

$$|GHZ\rangle = \frac{1}{\sqrt{2}}(|0\dots0\rangle + |1\dots1\rangle)$$

$$|W\rangle = \frac{1}{\sqrt{N}}(|10\dots00\rangle + |01\dots00\rangle + \dots + |00\dots01\rangle)$$

Требуется выполнить необходимые преобразования, чтобы точно различить эти два состояния. Возвращать 0, если первое состояние и 1, если второе.

Решение:

Для различения состояний достаточно измерить кубиты. Тогда, если было состояний GHZ, все кубиты будут в состоянии 0 или 1. Если же было состояние W, то только один кубит был в состоянии 1

В случае, когда $N = 1$, состояние не различить, так как в обоих случая может быть состояние $|1\rangle$

```
def get_last_state(state):
    res = state.measure()[0] # (outcome : str, state : post-measurement state)
    c = res.count('1')
    return 1 if c == 1 else 0
```

```
st = w
res = ''
if get_last_state(st) == 1: res = "W"
else: res = 'GHZ'
print(f'State detected: {res}')
```

2.3 Различение состояний 2 (2 балла)

Дано 2 кубита, которые могут быть в одном из двух состояний:

$$|S_0\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$$
$$|S_1\rangle = \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle)$$
$$|S_2\rangle = \frac{1}{2}(|00\rangle + |01\rangle - |10\rangle - |11\rangle)$$
$$|S_3\rangle = \frac{1}{2}(|00\rangle - |01\rangle - |10\rangle + |11\rangle)$$

Требуется выполнить необходимые преобразования, чтобы точно различить эти четыре состояния. Возвращать требуется индекс состояния (от 0 до 3).

Решение:

Заметим, что нулевое состояние получается применением оператора Адамара к обоим кубитам. Первое состояние получается из нулевого заменой во втором кубите плюса на минус, если этот второй кубит был равен единице. Аналогично второе состояние, только для первого кубита. Третье состояние получается применением замены для обоих кубитов (поэтому появляется минус там, где значения кубитов различны).

Такая замена происходит при применении к кубиту оператора Z.

Для различения нулевого состояния оператор Адамара и получим вектор $|00\rangle$ (обратное преобразование).

Для первого состояния нужно получить вектор $|01\rangle$. То есть проинвертировать второй кубит относительно состояние $|00\rangle$. Это можно сделать применив еще раз оператор Адамара к нему, так как $HZH = X$. К первому таже нужно применить оператор Адамара, чтобы вернуть в состояние $|0\rangle$

Аналогично для остальных состояний.

```
def get_state(state):
    qreg_q = QuantumRegister(2, 'q')
    creg_c = ClassicalRegister(2, 'c')
    circuit = QuantumCircuit(qreg_q, creg_c)
    circuit.initialize(state)
    circuit.h(qreg_q[0])
    circuit.h(qreg_q[1])

    cur = Statevector(circuit)
    res = cur.measure()[0] # (outcome : str, state : post-measurement state)

    num = int(res, 2)
    return num
```

```
get_state(S_0)
>>>        0
```

2.4 Написание оракула 1 (2 балла)

Требуется реализовать квантовый оракул на N кубитах ($1 \leq N \leq 8$), который реализует следующую функцию: $f(\boldsymbol{x}) = (\boldsymbol{bx}) \mod 2$, где $\boldsymbol{b} \in \{0,1\}^N$ вектор битов и $\boldsymbol{x}$ вектор кубитов. Выход функции записать в кубит $\boldsymbol{y}$. Количество кубитов N ($1 \leq N \leq 8$).

Заготовка для кода:

```
namespace Solution {
        open Microsoft.Quantum.Primitive;
        open Microsoft.Quantum.Canon;
        operation Solve (x : Qubit[], y : Qubit, b : Int[]) : ()
        {
            body
            {

            }
        }
}
```