ДЗ № 3: Машины Тьюринга и квантовые вычисления.

Сингин Александр. Группа А-13б-19.

1 Введение

2 Машины Тьюринга

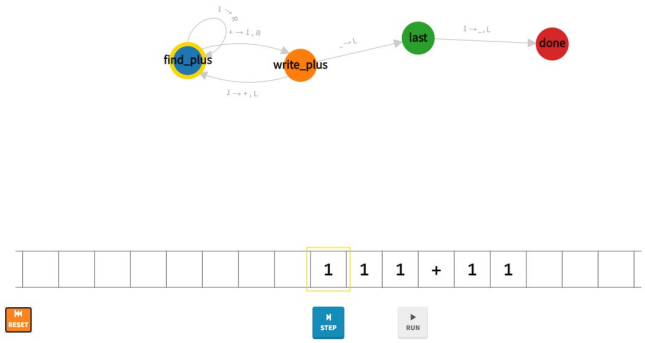
2.1 Операции с числами

1. Сложение двух унарных чисел

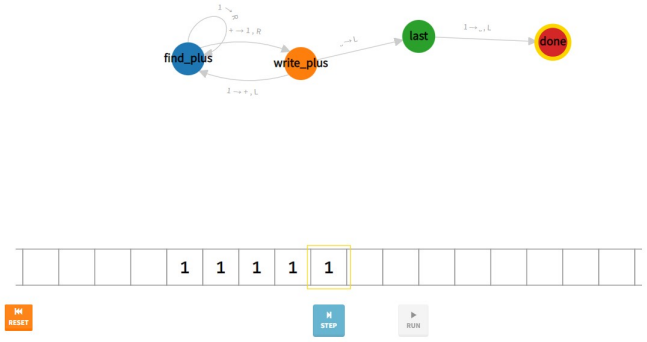
```
name: binary increment
source code: |+
  input: '111+11'
  blank: ' '
  start state: find_plus
  table:
    # Идем вправо до "+"
    find_plus:
      1: R
      +: {write: 1, R: write_plus}

    # Заменяем "+" на "1"
    write_plus:
      1: {write: +, L: find_plus}
      ' ': {L: last}

    # Удаляем посл. "1"
    last:
      1: {write: ' ', L: done}

    done:
positions:
  find_plus: {x: 275.17, y: 388.16, fixed: false}
  write_plus: {x: 366.02, y: 286.98}
  last: {x: 508.2, y: 247.79, fixed: false}
  done: {x: 672, y: 261.09}
```

Пример:

Введено: 111+11



Получено:



2. Умножение унарных чисел

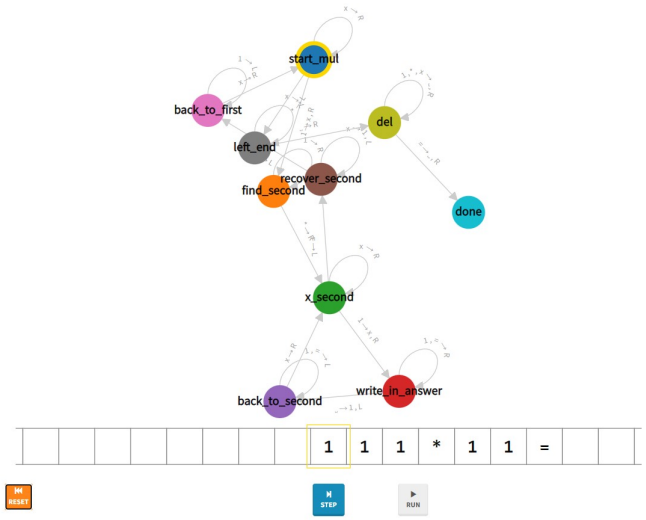
```
name: binary increment
source code: |-
  input: '111*11='
  blank: ' '
  start state: start_mul
  table:
    start_mul:
      1: {write: x, R: find_second}
      x: R
      '*': {L: left_end}

    # Идем ко 2-ому слагаемому
    find_second:
      1: R
      '*': {R: x_second}
```

```
    # Обработка 2-ого слагаемого
    x_second:
      x: R
      1: {write: x, R: write_in_answer}
      =: {L: recover_second}

    # Приписываем "1" в ответ
    write_in_answer:
      [1, =]: R
      ' ': {write: 1, L: back_to_second}

    # Возвращаемся ко 2-ому слагаемому
    back_to_second:
      [1, =]: L
      x: {R: x_second}

    # Восстанавливаем 2-ое слагаемое (Н-р, xx -> 11)
    recover_second:
      x: {write: 1, L}
      '*': {L: back_to_first}

    # Возвращаемся к 1-ому слагаемому; обр. остав. разряды 1-ого слагаемого
    back_to_first:
      1: L
      x: {R: start_mul}

    # Возвращаемся в начало
    left_end:
      x: L
      ' ': {R: del}

    # Удаляем входные данные
    del:
      [1, '*', x]: {write: ' ', R: del}
      =: {write: ' ', R: done}

    done:
positions:
  start_mul: {x: 375.1, y: 89.77, fixed: false}
```

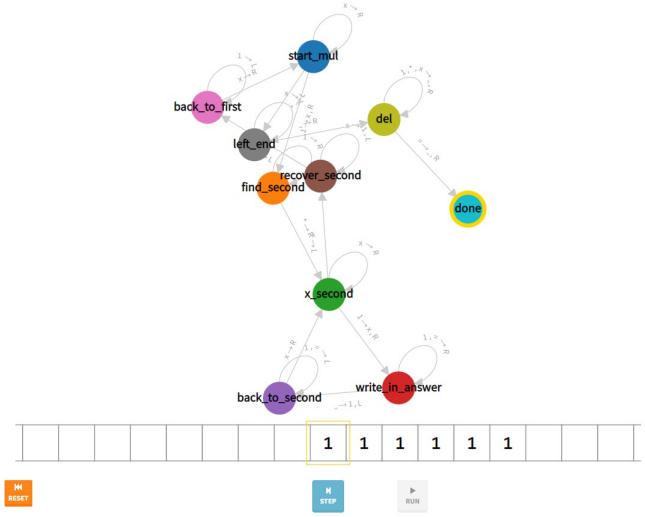
```
find_second: {x: 334.98, y: 249.53, fixed: false}
x_second: {x: 444.97, y: 339.81, fixed: false}
write_in_answer: {x: 503.66, y: 469.97, fixed: false}
back_to_second: {x: 360.58, y: 456.42, fixed: false}
recover_second: {x: 376.72, y: 203.31, fixed: false}
back_to_first: {x: 239.46, y: 122.74, fixed: false}
left_end: {x: 288.64, y: 197.28, fixed: false}
del: {x: 448.86, y: 162.93, fixed: false}
done: {x: 570, y: 250}
```

Пример:

Введено: 111*11=



Получено:



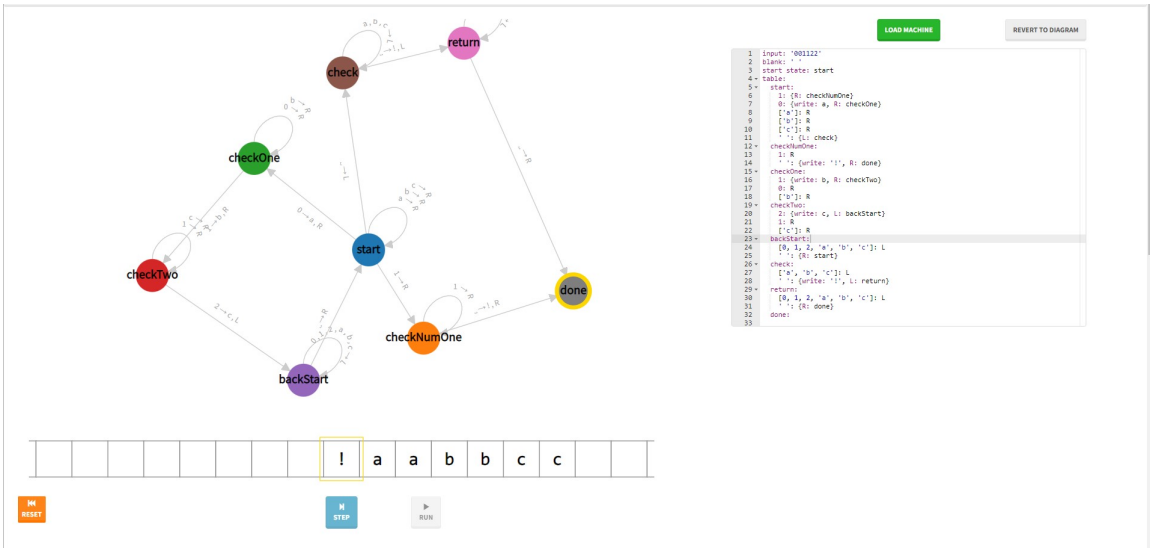
2.2 Операции с языками и символами

Реализуйте машины Тьюринга, которые позволяют выполнять следующие операции:

1. Принадлежность к языку $L = \{0^n 1^n 2^n\}, n \geq 0$

```
name: binary increment
source code: |
  input: '001122'
  blank: ' '
  start state: start
  table:
    start:
      1: {R: checkNumOne}
      0: {write: a, R: checkOne}
      ['a']: R
      ['b']: R
      ['c']: R
      ' ': {L: check}
    checkNumOne:
      1: R
      ' ': {write: '!', R: done}
    checkOne:
      1: {write: b, R: checkTwo}
      0: R
      ['b']: R
    checkTwo:
```

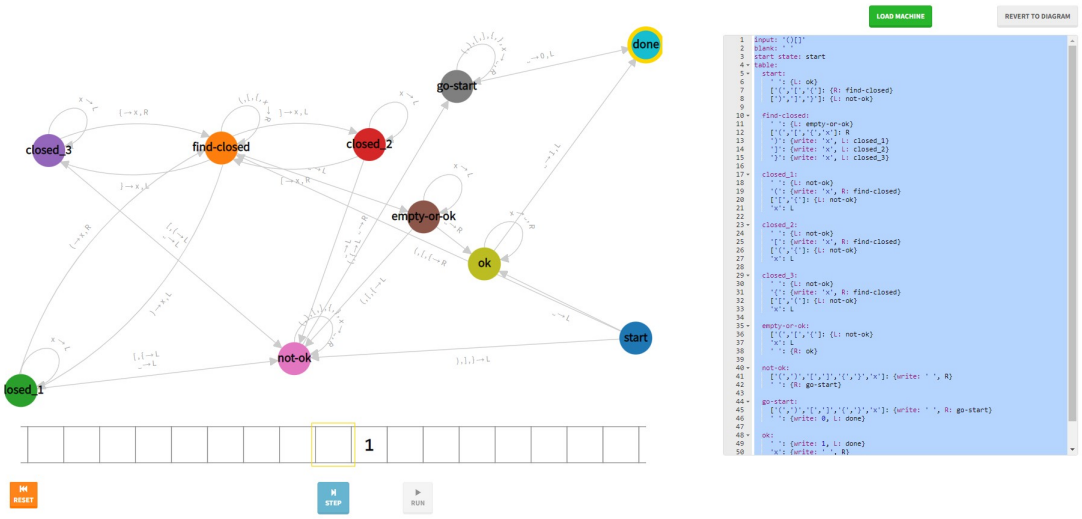
```
      2: {write: c, L: backStart}
      1: R
      ['c']: R
    backStart:
      [0, 1, 2, 'a', 'b', 'c']: L
      ' ': {R: start}
    check:
      ['a', 'b', 'c']: L
      ' ': {write: '!', L: return}
    return:
      [0, 1, 2, 'a', 'b', 'c']: L
      ' ': {R: done}
    done:
positions:
  start: {x: 432.84, y: 280.39}
  checkNumOne: {x: 499.75, y: 387.63}
  checkOne: {x: 293.72, y: 169.31}
  checkTwo: {x: 169.91, y: 311.54}
  backStart: {x: 353.72, y: 438.73}
  check: {x: 401.42, y: 65.42}
  return: {x: 548.8, y: 29.33}
  done: {x: 681.85, y: 330.26}
```



2. Проверка соблюдения правильности скобок в строке (Минимум 3 вида скобок).

```
input: '()[]'
blank: ' '
start state: start
table:
  start:
    ' ': {L: ok}
    ['(','[','{']: {R: find-closed}
    [')',']','}']: {L: not-ok}

  find-closed:
    ' ': {L: empty-or-ok}
    ['(','[','{','x']: R
    ')': {write: 'x', L: closed_1}
    ']': {write: 'x', L: closed_2}
    '}': {write: 'x', L: closed_3}

  closed_1:
    ' ': {L: not-ok}
    '(': {write: 'x', R: find-closed}
    ['[','{']: {L: not-ok}
    'x': L

  closed_2:
    ' ': {L: not-ok}
    '[': {write: 'x', R: find-closed}
    ['(','{']: {L: not-ok}
    'x': L

  closed_3:
    ' ': {L: not-ok}
    '{': {write: 'x', R: find-closed}
    ['[','(']: {L: not-ok}
    'x': L

  empty-or-ok:
    ['(','[','{']: {L: not-ok}
    'x': L
    ' ': {R: ok}
```

```
not-ok:
  ['(',')','[',']','{','}','x']: {write: ' ', R}
  ' ': {R: go-start}

go-start:
  ['(',')','[',']','{','}','x']: {write: ' ', R: go-start}
  ' ': {write: 0, L: done}

ok:
  ' ': {write: 1, L: done}
  'x': {write: ' ', R}

done:
```

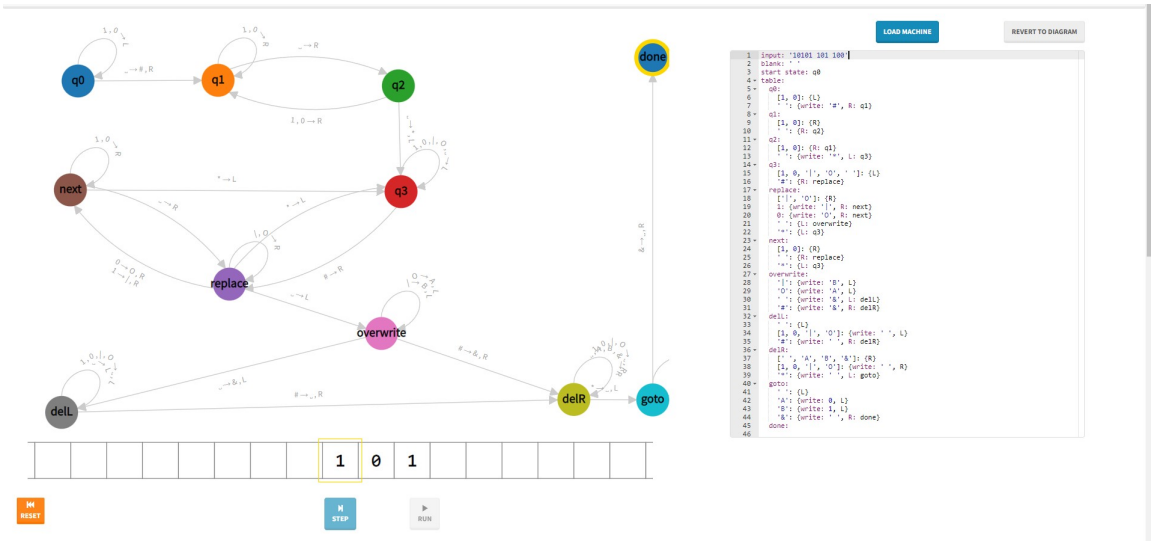


3. Поиск минимального по длине слова в строке (Слова состоят из символов 1 и 0 и разделены пробелом).

```
name: 'binary increment'
source code: |
  input: '10101 101 100'
  # input: '11 01 10'
  # input: '1 101 110'
  # input: '1'
  blank: ' '
  start state: q0
  table:
    q0:
      [1, 0]: {L}
      ' ': {write: '#', R: q1}
    q1:
      [1, 0]: {R}
      ' ': {R: q2}
    q2:
      [1, 0]: {R: q1}
      ' ': {write: '*', L: q3}
    q3:
      [1, 0, '|', 'O', ' ']: {L}
      '#': {R: replace}
    replace:
      ['|', 'O']: {R}
      1: {write: '|', R: next}
      0: {write: 'O', R: next}
      ' ': {L: overwrite}
      '*': {L: q3}
    next:
      [1, 0]: {R}
      ' ': {R: replace}
      '*': {L: q3}
    overwrite:
      '|': {write: 'B', L}
      'O': {write: 'A', L}
      ' ': {write: '&', L: delL}
      '#': {write: '&', R: delR}
    delL:
      ' ': {L}
```

```
      [1, 0, '|', 'O']: {write: ' ', L}
      '#': {write: ' ', R: delR}
    delR:
      [' ', 'A', 'B', '&']: {R}
      [1, 0, '|', 'O']: {write: ' ', R}
      '*': {write: ' ', L: goto}
    goto:
      ' ': {L}
      'A': {write: 0, L}
      'B': {write: 1, L}
      '&': {write: ' ', R: done}
    done:
positions:
  q0: {x: 81.08, y: 73.12}
  q1: {x: 251.31, y: 72.3}
  q2: {x: 470.63, y: 79.05}
  q3: {x: 473.86, y: 207.64}
  replace: {x: 265.19, y: 320.03}
  next: {x: 71.7, y: 205.53}
  overwrite: {x: 450.05, y: 380.05}
  delL: {x: 61.09, y: 475.98}
  delR: {x: 684.06, y: 460.28}
  goto: {x: 780, y: 460.8}
  done: {x: 780, y: 44.75}
editor contents: |
  input: '10101 101 100'
  blank: ' '
  start state: q0
  table:
    q0:
      [1, 0]: {L}
      ' ': {write: '#', R: q1}
    q1:
      [1, 0]: {R}
      ' ': {R: q2}
    q2:
      [1, 0]: {R: q1}
      ' ': {write: '*', L: q3}
    q3:
      [1, 0, '|', 'O', ' ']: {L}
```

```
  '#': {R: replace}
replace:
  ['|', 'O']: {R}
  1: {write: '|', R: next}
  0: {write: 'O', R: next}
  ' ': {L: overwrite}
  '*': {L: q3}
next:
  [1, 0]: {R}
  ' ': {R: replace}
  '*': {L: q3}
overwrite:
  '|': {write: 'B', L}
  'O': {write: 'A', L}
  ' ': {write: '&', L: delL}
  '#': {write: '&', R: delR}
delL:
  ' ': {L}
  [1, 0, '|', 'O']: {write: ' ', L}
  '#': {write: ' ', R: delR}
delR:
  [' ', 'A', 'B', '&']: {R}
  [1, 0, '|', 'O']: {write: ' ', R}
  '*': {write: ' ', L: goto}
goto:
  ' ': {L}
  'A': {write: 0, L}
  'B': {write: 1, L}
  '&': {write: ' ', R: done}
done:
```



3 Квантовые вычисления

Для выполнения заданий по квантовым вычислениям требуется QDK. Его можно скачать здесь: https://docs.microsoft.com/en-us/azure/quantum/install-overview-qdk.

Но можно использовать любой пакет, типа https://qiskit.org/.

В качестве решения задачи надо предоставить схему алгоритма для частного случая при фиксированном количестве кубитов и фиксированных состояниях.

3.1 Генерация суперпозиций 1 (1 балл)

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

$$S = \frac{1}{\sqrt{2}}(0 \dots 0 + \psi)$$

То есть требуется реализовать операцию, которая принимает на вход:

1. Массив кубитов q_s
2. Массив битов *bits* описывающих некоторое состояние ψ. Это массив имеет тот же самый размер, что и qs. Первый элемент этого массива равен 1.

Заготовка для кода:

```
namespace Solution {
        open Microsoft.Quantum.Primitive;
        open Microsoft.Quantum.Canon;
        operation Solve (qs : Qubit[]) : Int
```

```
        {
            body
            {

                return
            }
        }
}
```

Применяем к 1-ому кубиту оператор Адамара. Остальные к остальным кубитам, если они равны 1 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)
```

3.2 Различение состояний 1 (1 балл)

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

$$GHZ = \frac{1}{\sqrt{2}}(0\ldots0 + 1\ldots1)$$

$$W = \frac{1}{\sqrt{N}}(10\ldots00 + 01\ldots00 + \cdots + 00\ldots01)$$

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

Заготовка для кода:

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

            }
        }
}
```

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 $|1\rangle$

```
N = int(input('N = '))
ghz = [1/math.sqrt(2)]
for i in range (1, 2**N-1):
    ghz.append(0)
ghz.append(1/math.sqrt(2))
ghz = Statevector(ghz)
ghz.draw('latex')
w = [0]
for i in range (1, 2**N):
    if i & i-1:
        w.append(0)
    else:
        w.append(1/math.sqrt(N))
w = Statevector(w)
w.draw('latex')
def Solve(state):
    res = state.measure()[0]
    c = res.count('1')
    return 1 if c == 1 else 0
res = ''
if Solve(w) == 1:
    res = "W"
else: res = 'GHZ'
print(f'Ответ: {res}')
```