ТМВ ДЗ №3

А-13б-19 Оленичев Владимир

1. Машины Тьюринга

1.1 Операции с языками и символами

Реализуйте машины Тьюринга, которые позволяют выполнять следующие операции:

1. **Сложение двух унарных чисел (1 балл)**

   Алгоритм:

   Движемся вправо, пока не встретили '+'. Заменяем '+' на 1 и движемся в конец. Находим конечную единицу и удаляем её. Движемся к голове.

```
input: 1111+11
blank: ' '
start state: start

table:

start:
    1: R
    '+': {write: '1', R: go_tail}
go_tail:
    1: R
    ' ': {L: del}
del:
     1: {write: ' ', L: go_head}

go_head:
      1: L
      ' ': {R: fin}
 fin:
```

2. **Умножение унарных чисел (1 балл)**

   Копируем за знак '=' единицы первого множителя столько раз, сколько единиц во втором.

   Алгоритм:

   Движемся в конец выражения, ставим знак '=' и возвращаемся в начало. Переносим второй множитель за знак равенства столько раз, сколько единиц в первом. После каждого переноса восстанавливаем замененный на промежуточные символы второй множитель. Промежуточный символ 'x'. В конце удаляем левую часть выражения, в т.ч. знак '='.

```
input: '11*111'
blank: ' '
start state: set_eq
table:

  set_eq:
        [1,'*']: R
```

```
        ' ': {write: '=', L: go_head}

    go_head:
            [1,'*']: L
            [' ', x]: {R: go_mul}

    go_mul:
          1: {write: x, R: go_second}
          x: R
          '*': {L: go_first}

    go_second:
          1: R
          '*': {R: second}

    second:
          1: {write: x, R: to_ans}
          x: R
          =: {L: res_second}

    to_ans:
          [1, =]: R
          ' ': {write: 1, L: back_second}

    back_second:
          [1, =]: L
          x: {R: second}

    res_second:
          x: {write: 1, L}
          '*': {L: go_head}

    go_first:
          x: L
          ' ': {R: clear}

    clear:
          [1,'*',x]: {write: ' ', R}
          =: {write: ' ', R: fin}
    fin:
```

1.2 Операции с языками и символами

Реализуйте машины Тьюринга, которые позволяют выполнять следующие операции:

1. Принадлежность к языку $L = \{0^n 1^n 2^n\}, n \geq 0$ (0.5 балла)

   Алгоритм:

   Все первые вхождения 0, 1, 2 заменяем на 'x'. Возвращаемся в начало. Повторяем предыдущий шаг, пока слово не будет заменено на все 'x' (иначе слово не принадлежит языку). 1: слово принадлежит языку 0: нет. Пустое слово тоже принадлежит языку.

```
input: '001122'
blank: ' '
start state: start


table:
```

```
  start:
    ' ': {L: good_word}
    0: {write: X, R: go_one}
    [1, 2]: {R: bad_word}
    X: R

  go_one:
    1: {write: X, R: go_two}
    [' ', 2]: {R: bad_word}
    [X, 0]: R

  go_two:
    2: {write: X, L: go_head}
    ' ': {R: bad_word}
    [X, 1]: R

  go_head:
    ' ': {R: start}
    [0,1,2,X]: L

  good_word:
    ' ': {write: 1, L: to_fin}
    X: {write: ' ', L}

  bad_word:
    ' ': {L: clear}
    [0, 1, 2, X]: R

  clear:
    ' ': {write: 0, L: to_fin}
    [0, 1, 2, X]: {write: ' ', L}

  to_fin:
    ' ': {R: fin}

  fin:
```

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

   Алгоритм:

   Ищем первую закрывающую скобку. Заменяем её на 'x'. Возвращаемся в начало. Ищем открывающую скобку такого же вида. Заменяем её на 'x'. 1: слово принадлежит языку (все 'x'), 0: нет Пустое слово - правильная скобочная последовательность.

```
input: '(([{}]))'
blank: ' '
start state: start


table:
  start:
    ' ': {L: ok}
    ['(', '[', '{']: {R: search_rb}
    [')', ']', '}']: {L: bad}

  search_rb:
    ' ': {L: eol}
    ['(', '[', '{', 'x']: R
```

```
    ')': {write: 'x', L: rb_1}
    ']': {write: 'x', L: rb_2}
    '}': {write: 'x', L: rb_3}

  rb_1:
    ' ': {R: bad}
    '(': {write: 'x', R:  search_rb}
    ['[', '{']: {L: bad}
    'x': L

  rb_2:
    ' ': {R: bad}
    '[': {write: 'x', R: search_rb}
    ['(', '{']: {L: bad}
    'x': L

  rb_3:
    ' ': {R: bad}
    '{': {write: 'x', R: search_rb}
    ['[', '(']: {L: bad}
    'x': L

  eol:
    ['(', '[', '{']: {L: bad}
    'x': L
    ' ': {R: ok}

  bad:
    ['(', ')', '[', ']', '{', '}', 'x']: {write: ' ', R}
    ' ': {R: go_head}

  go_head:
    ['(', ')', '[', ']', '{', '}', 'x']: {write: ' ', R: go_head}
    ' ': {write: 0, L: fin}

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

  fin:
```

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

   Алгоритм:

   Обрабатываем первые 2 слова, сравниваем их длину. Если первое слово больше, стираем его и восстанавливаем второе слово из промежуточных 'a' и 'b'. Если второе больше, то копируем на его место первое, после удаляем остатки. Повторяем до конца строки.

```
input: '1010 010 1'
blank: ' '
start state: first_word
table:

  first_word:
    0: {write: a, R: to_second}
    1: {write: b, R: to_second}
    [a, b]: R
```

```
  ' ': {L: first_is_small}

to_second:
  [0, 1]: R
  ' ': {R: second_word}

second_word:
  ' ': {L: one_left}
  0: {write: a, L: to_first}
  1: {write: b, L: to_first}
  [a, b]: {R: second_not_null}

second_not_null:
  [a, b]: R
  0: {write: a, L: to_first}
  1: {write: b, L: to_first}
  ' ': {L: second_is_small}

to_first:
  [a, b]: L
  ' ': {L: to_begin_first}

to_begin_first:
  [0, 1, a, b]: L
  ' ': {R: first_word}

one_left:
  ' ': {L: restore_and_exit}

restore_and_exit:
  a: {write: 0, L}
  b: {write: 1, L}
  [0, 1]: L
  ' ': {R: fin}

# первое меньше, замена 2-го на 1-е
first_is_small:
  [a, b]: L
  ' ': {R: restore_first}

restore_first:
  a: {write: 0, R}
  b: {write: 1, R}
  ' ': {R: cut_second}

cut_second:
  [a, b, 0, 1]: {write: a, R}
  ' ': {L: return_and_copy}

return_and_copy:
  a: L
  ' ': {L: copy_first}

copy_first:
  [a, b]: L
  0: {write: a, R: carry0}
```

```
  1: {write: b, R: carry1}
  ' ': {R: delete_to_word}

carry0:
  [a, b]: R
  ' ': {R: carry0_in_second}

carry0_in_second:
  a: R
  [0, 1, ' ']: {L: set0_and_return}

set0_and_return:
  a: {write: 0, L: return_and_copy}
  ' ': {L: return_and_copy}

carry1:
  [a, b]: R
  ' ': {R: carry1_in_second}
carry1_in_second:
  a: R
  [0, 1, ' ']: {L: set1_and_return}
set1_and_return:
  a: {write: 1, L: return_and_copy}
  ' ': {L: return_and_copy}

delete_to_word:
  [a, b]: {write: ' ', R}
  [0, 1]: {L: to_begin_first}
  ' ': {R: delete_to_word_in_sec}

delete_to_word_in_sec:
  [a, b]: {write: ' ', R}
  [0, 1]: {L: to_begin_first}
  ' ': {R: fin}

# 2-е меньше
second_is_small:
  [a, b]: L
  ' ': {L: to_begin_first_and_del}

to_begin_first_and_del:
  [0, 1, a, b]: L
  ' ': {R: delete_first}

delete_first:
  [0, 1, a, b]: {write: ' ', R}
  ' ': {R: restore_second}

restore_second:
  a: {write: 0, R}
  b: {write: 1, R}
  ' ': {L: to_begin_first}
fin:
```

2 Квантовые вычисления

2.1 Генерация суперпозиций 1 (1 балл)

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

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

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 q_s. Первый элемент этого массива равен 1.

Решение:

В начале у нас есть N незаисимых кубитов $|0\rangle$ Первые кубиты векторов различны, применим оператор Адамара к первому кубиту Все кубиты qs равны $0 \Rightarrow$ если кубит $bits[i] = 1$, то нужно запутать i-ый кубит, а если кубит $bits[i] = 0$, то не нужно, т.к кубиты совпадают и равны 0.

Прога

```
namespace Solution {
    open Microsoft.Quantum.Primitive;
    open Microsoft.Quantum.Canon;
    operation Solve (qs : Qubit[], bits : Bool[]) : Unit
    {
        body
        {
            H(qs[0]);
            for i in 1..Length(qs) - 1 {
                if (bits[i]) {
                    CNOT(qs[0], qs[i]);
                }
            }
        }
    }
}
```

2.2 Различение состояний 1 (1 балл)

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

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

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

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

Решение:

Чтобы измерить состояние системы надо измерить кубиты При $N > 1$ состояние 1: N нулей, либо N единиц, состояние 2: 1 единица При $N = 1$ состояния не различить (в обоих состояниях может выпасть вектор, который содержит одну единицу)

Прога

```
namespace Solution {
    open Microsoft.Quantum.Primitive;
    open Microsoft.Quantum.Canon;
    operation Solve (qs : Qubit[]) : Int
    {
        body
        {
            mutable ones = 0;
            for i in 0..Length(qs) - 1 {
                if (M(qs[i]) == One) {  // measurement
                    set ones += 1;
                }
            }
            if (ones == 1) {
                return 1;
            }
            return 0;
        }
    }
}
```