# Теоретические модели вычислений

# Машины Тьюринга и квантовые вычисления

*Иванов Олег А-13а-19*

# Машины Тьюринга

Машины построены с возвратом без сохранения.

## 1.1) Сложение двух унарных чисел

Входной формат:

```
<число>+<число>
```

Для сложения достаточно заменить + на 1.
Код МТ:

```
# 1.1) Сложение двух унарных чисел
input: '111+111'
blank: ' '
start state: right
table:
  # движение вправо
  right:
    [1,+]: R
    ' '  : {L: del}
  # удаляем последнюю единицу
  del:
    1: {write: ' ', L: del_plus}
  # записываем "удалённую" единицу вместо +
  del_plus:
    1: L
    +: {write: 1, L: left}
  # движение влево
  left:
    1: L
    ' ': {R: done}
  done:
```

## 1.2) Умножение двух унарных чисел

Входной формат:

```
<число>*<число>
```

Переносим первый множитель за знак равенства столько раз, сколько единиц во втором.
Промежуточный символ: a; в конце стираем данные, не относящиеся к ответу.
Код МТ:

```
# 1.2) Умножение двух унарных чисел
input: '11*11'
blank: ' '
start state: put_eq
table:
  # поставить в конец знак равенства
  put_eq:
    [1, '*']: R
    ' ': {write: =, L: left}
  # движение влево
  left:
    [1, '*']: L
    ' ': {R: start_mul}
  # начать умножение
  start_mul:
    1: {write: a, R: to_second}
    a: R
    '*': {L: to_left_end}
  # движение ко второму множителю
  to_second:
    1: R
    '*': {R: second}
  # обработка второго множителя
  second:
    a: R
    1: {write: a, R: carry_to_answer}
    =: {L: restore_second}
  # перенос единицы в ответ
  carry_to_answer:
    [1, =]: R
    ' ': {write: 1, L: back_to_second}
  #возврат ко второму множителю
  back_to_second:
    [1, =]: L
    a: {R: second}
  # восстановить второй множитель (обработка закончена)
  restore_second:
    a: {write: 1, L}
```

```
    '*': {L: to_first}
  # перейти к необработанному разряду
  to_first:
    1: L
    a: {R: start_mul}
  # к левому краю
  to_left_end:
    a: L
    ' ': {R: del_all}
  # очистить входные данные
  del_all:
    [1, '*', a]: {write: ' ', R: del_all}
    =: {write: ' ', R: done}
  done:
```

## 2.1) Принадлежность к языку $L = \{0^n1^n2^n\}, n \geq 0$

Результат работы: d, если успех; f, если нет.

Будем строго по очереди убирать символы 0, 1 и 2.

Если нарушается порядок, то ошибка.

Код МТ:

```
# 2.1) Принадлежность к языку L = {0^n1^n2^n}, n>=0
input: '001122'
blank: ' '
start state: zero
table:
  # взятие нуля
  zero:
    ' ': {L: done}
    0: {write: a, R: one}
    [1, 2]: {R: fail}
    a: R
  # поиск и взятие единицы
  one:
    [0, a]: R
    1: {write: a, R: two}
    [2, ' ']: {L: fail}
  # поиск и взятие двойки
  two:
    [1, a]: R
    2: {write: a, L: back_zero}
    ' ': {L: fail}
  # возврат ко взятию нуля
  back_zero:
    [a, 2, 1, 0]: L
    ' ': {R: zero}
  # успех
```

```
  done:
    a: {write: ' ', L}
    ' ': {write: d, R: exit}
  # неудача
  fail:
    [a, 0, 1, 2]: R
    ' ': {L: del}
  # стереть
  del:
    [a, 0, 1, 2]: {write: ' ', L}
    ' ': {write: f, R: exit}
  exit:
    ' ': L
```

## 2.2) Проверка соблюдения правильности скобок в строке (минимум 3 вида скобок)

Результат работы: d, если успех; f, если нет.

Виды скобок: (), [], {}

Правильной скобочной последовательностью считаем стоящие рядом скобки одного вида (либо разделённые дополнительными символами).

Ищем первую правую скобку, затем смотрим предшествующую ей скобку: если она того же вида, удаляем обе; иначе ошибка.

Код МТ:

```
# 2.2) Проверка соблюдения правильности скобок в строке (минимум 3 вида скоб
# <тривиальная правильная последовательность> ::= () | [] | {} |
input: '([{}])({[]}){}'
#input: '([{}]'
#input: ')'
#input: '('
#input: '()[]{}'
#input: ''
blank: ' '
start state: left
table:
  # смотрим слева
  left:
    ' ': {L: to_begin}
    a: R
    '(': {R: left_r}
    '[': {R: left_sq}
    '{': {R: left_fig}
    ')': {write: a, L: right_r}
    ']': {write: a, L: right_sq}
    '}': {write: a, L: right_fig}
  # встречена левая круглая скобка
```

```
left_r:
  ' ': {L: fail}
  a: R
  '(': {R: left_r}
  '[': {R: left_sq}
  '{': {R: left_fig}
  ')': {write: a, L: right_r}
  ']': {write: a, L: right_sq}
  '}': {write: a, L: right_fig}
# встречена левая квадратная скобка
left_sq:
  ' ': {L: fail}
  a: R
  '(': {R: left_r}
  '[': {R: left_sq}
  '{': {R: left_fig}
  ')': {write: a, L: right_r}
  ']': {write: a, L: right_sq}
  '}': {write: a, L: right_fig}
# встречена левая фигурная скобка
left_fig:
  ' ': {L: fail}
  a: R
  '(': {R: left_r}
  '[': {R: left_sq}
  '{': {R: left_fig}
  ')': {write: a, L: right_r}
  ']': {write: a, L: right_sq}
  '}': {write: a, L: right_fig}
# встречена правая круглая скобка
right_r:
  a: L
  '(': {write: a, L: try_left}
  ['{', '[']: {write: a, L: fail}
  ' ': {R: fail}
# встречена правая квадратная скобка
right_sq:
  a: L
  '[': {write: a, L: try_left}
  ['{', '(']: {write: a, L: fail}
  ' ': {R: fail}
# встречена правая фигурная скобка
right_fig:
  a: L
  '{': {write: a, L: try_left}
  ['(', '[']: {write: a, L: fail}
  ' ': {R: fail}
# посмотреть слева
try_left:
  ' ': {R: left}
```

```
    [a, '{', '[', '(']: {R: back_to_left}
  # перейти влево
  back_to_left:
    a: {L: left}
    ' ': L
  # перейти в начало слова (при успехе)
  to_begin:
    ' ': {R: done}
    a: L
    ['{', '[', '(']: {L: try_left}
  # успех
  done:
    [a, ' ']: {write: d, R: to_end}
  # перейти в конец слова
  to_end:
    [a, '{', '[', '(', '}', ']', ')']: R
    ' ': {L: clear}
  # стереть все символы из исходного алфавита
  clear:
    [a, '{', '[', '(', '}', ']', ')']: {write: ' ', L}
    d: L
    ' ': {R: exit}
  # неудача
  fail:
    [a, '{', '[', '(', '}', ']', ')', ' ']: {R: to_end}
  # выход
  exit:
    ' ': {write: f, R: back_to_left}
```

## 2.3) Поиск минимального по длине слова в строке (слова состоят из символов 1 и 0 и разделены пробелом

Обрабатываются первые два слова: сравнивается их длина.

Если первое слово оказалось больше, стираем его и восстанавливаем второе из промежуточных символов (0 мы заменяли на a, 1 на b).

Если второе оказалось больше, копируем восстановленное первое на место второго и удаляем остатки.

Код МТ:

```
# 2.3) Поиск минимального по длине слова в строке
# (слова состоят из символов 1 и 0 и разделены пробелом)
input: '1010 010 01 000'
blank: ' '
start state: first_word
table:
  # обрабатываем первое слово
  # можем установить, меньше ли оно второго
```

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

# движение ко второму слову
to_second:
  [0, 1]: R
  ' ': {R: second_word}
# обрабатываем второе слово
# можем установить, меньше ли оно первого
second_word:
  ' ': {L: one_left}
  0: {write: a, L: to_first}
  1: {write: b, L: to_first}
  [a, b]: {R: second_not_null}
# для корректного завершения
second_not_null:
  [a, b]: R
  0: {write: a, L: to_first}
  1: {write: b, L: to_first}
  ' ': {L: second_is_small}

# к первому слову
to_first:
  [a, b]: L
  ' ': {L: to_begin_first}
# к началу первого слова
to_begin_first:
  [0, 1, a, b]: L
  ' ': {R: first_word}
# шаг влево для завершения
one_left:
  ' ': {L: restore_and_exit}
# восстановить значение и завершить
restore_and_exit:
  a: {write: 0, L}
  b: {write: 1, L}
  [0, 1]: L
  ' ': {R: done}
#-----------------------------------
# первое слово меньше, заменим им второе
first_is_small:
  [a, b]: L
  ' ': {R: restore_first}
# восстановить слово
restore_first:
  a: {write: 0, R}
  b: {write: 1, R}
```

```
  ' ': {R: cut_second}
# заполнить второе слово (для корректного копирования)
cut_second:
  [a, b, 0, 1]: {write: a, R}
  ' ': {L: return_and_copy}
# вернуться к первому слову и начать копирование
return_and_copy:
  a: L
  ' ': {L: copy_first}
# копирование первого
copy_first:
  [a, b]: L
  0: {write: a, R: carry0}
  1: {write: b, R: carry1}
  ' ': {R: delete_to_word}
# перенести 0 для второго слова
carry0:
  [a, b]: R
  ' ': {R: carry0_in_second}
# перенести 0 внутри второго слова
carry0_in_second:
  a: R
  [0, 1, ' ']: {L: set0_and_return}
# записать 0, вернуться и продолжить копирование
set0_and_return:
  a: {write: 0, L: return_and_copy}
  ' ': {L: return_and_copy}
# аналогично для 1
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}
# удалить остатки первого слова
delete_to_word:
  [a, b]: {write: ' ', R}
  [0, 1]: {L: to_begin_first}
  ' ': {R: delete_to_word_in_sec}
# удалить остатки второго слова
delete_to_word_in_sec:
  [a, b]: {write: ' ', R}
  [0, 1]: {L: to_begin_first}
  ' ': {R: done}

#-----------------------------------
# второе слово оказалось меньше
```

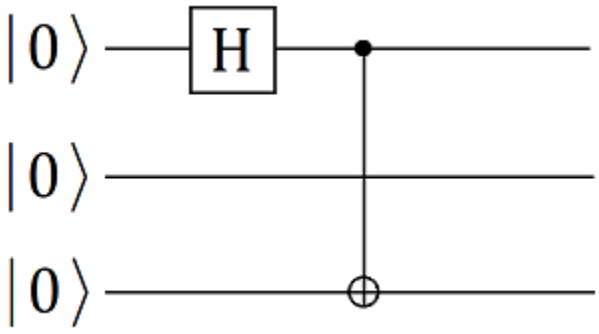
```
  second_is_small:
    [a, b]: L
    ' ': {L: to_begin_first_and_del}
  # перейти к началу первого слова и удалить его
  to_begin_first_and_del:
    [0, 1, a, b]: L
    ' ': {R: delete_first}
  # удаление
  delete_first:
    [0, 1, a, b]: {write: ' ', R}
    ' ': {R: restore_second}
  # восстановление бывшего второго слова (оно теперь первое)
  restore_second:
    a: {write: 0, R}
    b: {write: 1, R}
    ' ': {L: to_begin_first}
  done:
```

# Квантовые вычисления

## 3.1) Генерация суперпозиций 1

Воспользуемся алгоритмом для генерации суперпозиции двух базисных состояний, одним из которых будет нулевой вектор.
В этом случае алгоритм упрощается:
применяем оператор Адамара к первому кубиту (по условию, во втором векторе он единичный $\Rightarrow$ первый различающийся);
затем спутываем его со всеми единичными кубитами второго вектора.
Частный случай:

$$|S\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |101\rangle)$$




Код Q#:

```
operation Diff (qs : Qubit[], bits : Bool[]) : Unit
    {
```

```
        H(qs[0]); //по условию первый бит всегда 1 == различие в первом
        for i in 1 .. Length(qs)-1{
            if(bits[i]){            //этот бит отличается от нулевого
                CNOT(qs[0], qs[i]); //запутываем
            }

        }
    }
```

## 3.2) Различение состояний 1

Рассмотрим случай при $N = 3$:

$$|GHZ\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$$

$$|W\rangle = \frac{1}{\sqrt{3}}(|100\rangle + |010\rangle + |001\rangle)$$

Для различения достаточно измерить кубиты: если они были в первом состоянии, при измерении получим либо $N$ нулей, либо $N$ единиц, иначе ровно одну единицу.
Однако для $N = 1$ имеем:

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

$$|W\rangle = |1\rangle$$

В этом случае мы не можем точно различить состояния, так как нет преобразования, сохраняющего длины и углы, которое позволит различить $|+\rangle$ и $|1\rangle$.
Код Q#:

```
operation States (qs : Qubit[]) : Int
    {
        mutable ones = 0;
        for i in 0 .. Length(qs)-1{
            if(M(qs[i]) == One){
                set ones += 1;
            }
        }
        if(ones == Length(qs) or ones == 0){
            return 0;
        }
        else {
            return 1;
        }
    }
```

## 3.3) Различение состояний 2

Для различения этих четырёх состояний применим оператор Адамара к каждому кубиту. Он будет иметь вид:

$$H^{\otimes 2} = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$$

Представим указанные состояния в векторном виде:

$$|S_0\rangle = \frac{1}{2}\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, |S_1\rangle = \frac{1}{2}\begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \end{pmatrix}, |S_2\rangle = \frac{1}{2}\begin{pmatrix} 1 \\ 1 \\ -1 \\ -1 \end{pmatrix}, |S_3\rangle = \frac{1}{2}\begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \end{pmatrix}$$

Теперь применим оператор Адамара к этим векторам и проанализируем результат:

$$H^{\otimes 2}|S_i\rangle = |S_i'\rangle, 0 \leq i \leq 3$$

$$|S_0'\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, |S_1'\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, |S_2'\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, |S_3'\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Мы получили легкоразличимые состояния: $|00\rangle$, $|01\rangle$, $|10\rangle$ и $|11\rangle$. То есть, достаточно измерить кубиты, откуда тривиально будет следовать ответ.
Код Q#:

```
operation States2 (qs : Qubit[]) : Int
    {
        H(qs[0]);
        H(qs[1]); //тензорный H2
        if(M(qs[0]) == Zero){
            if(M(qs[1]) == Zero){
                return 0;  // |00>
            }
            else {
                return 1;  // |01>
            }
        }
        else{
            if(M(qs[1]) == Zero){
                return 2;  // |10>
            }
            else {
                return 3;  // |11>
            }
        }
    }
```