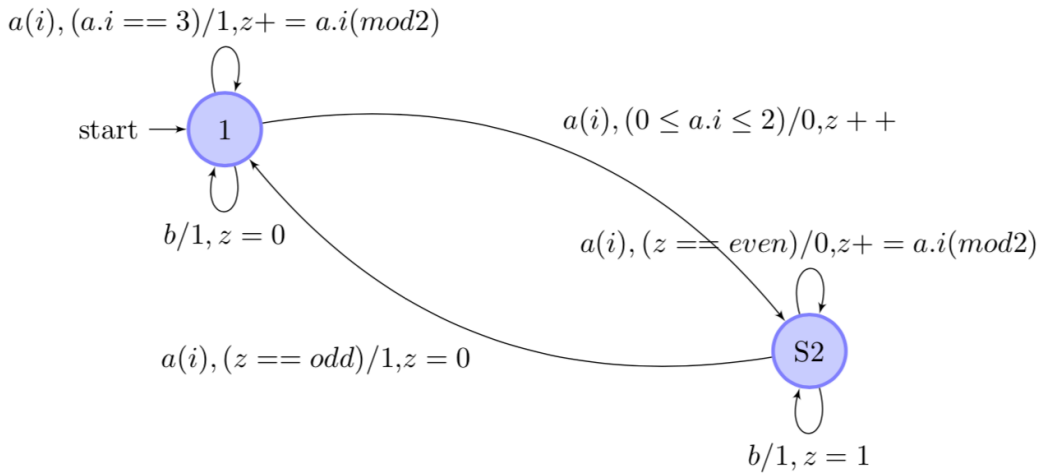
1. Введение

В рамках данной лабораторной работы был программно реализован данный расширенный автомат.



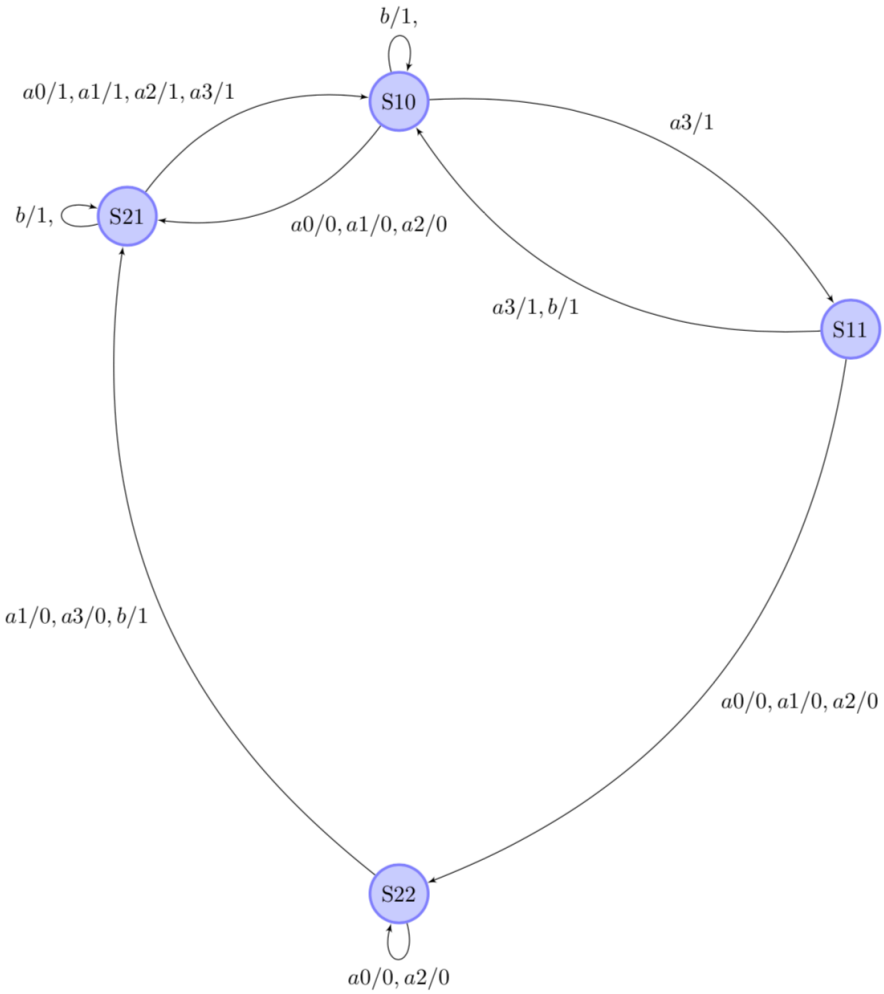
Затем был сгенерирован набор тестов («проверяющий тест») по этой модели, с помощью которого было проведено тестирование программы, и наконец было проведено мутационное тестирование для проверки полноты набора тестов.

2. Формальное описание системы (вид автомата, его характеристики).

Был построен детерминированный конечный автомат. Система представлена автоматом с 4 состояниями и 20 переходами.

Состояния:

- **S10** – состояние 1 (z=0);
- **S11** – состояние 1 (z==odd);
- **S21** – состояние 2 (z==odd);
- **S22** – состояние 2 (z==even).



3. Описание реализации (язык программирования, способ реализации часов в системе)

Автомат из раздела 1 было реализован программно, исходный код программной реализации на ЯП Python в Приложении 1.

4. Характеристики построенного теста, формат подачи теста

Для генерации тестовых последовательностей был использован инструмент fsm (https://github.com/kitidis/ fsm) в режиме черного ящика / W-метода. Текстовое описание автомата приведено в Приложение 2. В результате генерации тестов были получены следующие тестовые последовательности:

```
B/1 A0/0 A0/1
B/1 A3/1
A0/0 B/1 B/1 A0/1
A0/0 B/1 A0/1 B/1 A0/0 A0/1
A0/0 B/1 A0/1 B/1 A3/1
A0/0 B/1 A0/1 A0/0 A0/1
A0/0 B/1 A0/1 A0/0 A3/1
A0/0 B/1 A0/1 A1/0 A0/1
A0/0 B/1 A0/1 A2/0 A0/1
A0/0 B/1 A0/1 A2/0 A3/1
A0/0 B/1 A0/1 A3/1 A0/0 A0/0
A0/0 B/1 A0/1 A3/1 A3/1
A0/0 B/1 A1/1 A0/0
A0/0 B/1 A1/1 A3/1
A0/0 B/1 A2/1 A0/0
A0/0 B/1 A2/1 A3/1
A0/0 B/1 A3/1 A0/0 A0/1
A0/0 B/1 A3/1 A3/1
A0/0 A0/1 A0/0 A0/1
A0/0 A0/1 A3/1
A0/0 A1/1 A0/0 A0/1
A0/0 A1/1 A3/1
A0/0 A2/1 A0/0 A0/1
A0/0 A2/1 A3/1
A0/0 A3/1 A0/0
A1/0 A0/1
A2/0 A0/1
A3/1 A0/0
```

5. Описание процедуры генерации мутантов

С использованием пакета **MutPy 0.5.1** для программной реализации из раздела 3 были сгенерированы мутанты. Результаты запуска генерации мутантов представлены в Приложение 3.
Модификация программы влияет на работу автомата и содержит отклонения от спецификации. Если ошибка была обнаружена с помощью тестов, то мутант считается "убитым" (*killed*).
При этом были генерированы мутанты с различными операторами мутаций:

- ROR - замена оператора отношений;
- ASR - замена оператора присваивания;
- COI - добавление условного оператора;
- CRP - замена константы;

- LCR - замена логического оператора.

6. Результаты тестирования (в виде полноты теста относительно множества построенных мутантов)

Программная реализация при мутационном тестировании показала следующие результаты:

Результат	Количество	Доля
Killed	49	90.74%
Survived	5	9.26%
Incompitent	0	0.00%
Timeout	0	0.00%
Всего мутантов:	54	100%

Таким образом, полнота равна 90.74%.
Выжившие разделены на две группы: мутанты, которые не относятся к автоматной модели и те, которые относятся к петлям по значению z, которые не достижимы тестовыми последовательностями длины 4.

```
  - [#   3] ASR avtomat:
--------------------------------------------------------------
  29:         if inp == Value.B1:
  30:             self.z = 0
  31:             return 1
  32:         if inp == Value.A3:
- 33:             self.z += 1
+ 33:             self.z -= 1
  34:             return 1
  35:         if (inp == Value.A0 or inp == Value.A1 or inp ==
Value.A2):
  36:             self.z += 1
  37:             self.state = States.Q
--------------------------------------------------------------
[0.00956 s] survived
  - [#   4] ASR avtomat:
--------------------------------------------------------------
  32:         if inp == Value.A3:
  33:             self.z += 1
  34:             return 1
  35:         if (inp == Value.A0 or inp == Value.A1 or inp ==
Value.A2):
- 36:             self.z += 1
+ 36:             self.z -= 1
  37:             self.state = States.Q
```

```
  38:             return 0
  39:
  40:     def _q(self, inp):
------------------------------------------------------------
[0.00665 s] survived
   - [#   5] ASR avtomat:
------------------------------------------------------------
  42:             self.z = 1
  43:             return 1
  44:         if self.z % 2 == 0:
  45:             if (inp == Value.A1 or inp == Value.A3):
- 46:                 self.z += 1
+ 46:                 self.z -= 1
  47:             return 0
  48:         if self.z % 2 == 1:
  49:             self.z = 0
  50:             self.state = States.P
------------------------------------------------------------
[0.00686 s] survived
  - [#  19] CRP avtomat:
------------------------------------------------------------
   5:     A0 = 0
   6:     A1 = 1
   7:     A2 = 2
   8:     A3 = 3
-  9:     B1 = 4
+  9:     B1 = 5
  10:
  11:
  12: class States(Enum):
  13:     P = 0
------------------------------------------------------------
[0.01052 s] survived
   - [#  21] CRP avtomat:
------------------------------------------------------------
  10:
  11:
  12: class States(Enum):
  13:     P = 0
- 14:     Q = 1
+ 14:     Q = 2
  15:
  16:
  17: class System:
  18:
------------------------------------------------------------
[0.00750 s] survived
```

Приложение 1. Программная реализация расширенного автомата

avtomat.py

```
from enum import Enum


class Value(Enum):
    A0 = 0
    A1 = 1
    A2 = 2
    A3 = 3
    B1 = 4


class States(Enum):
    P = 0
    Q = 1


class System:
    __VALUE_INITIAL = 0
    __STATES_INITIAL = States.P

    def _input(self, var):
        if self.state == States.P:
            return self._p(var)
        if self.state == States.Q:
            return self._q(var)

    def _p(self, inp):
        if inp == Value.B1:
            self.z = 0
            return 1
        if inp == Value.A3:
            self.z += 1
            return 1
        if inp == Value.A0 or inp == Value.A1 or inp == Value.A2:
            self.z += 1
            self.state = States.Q
            return 0

    def _q(self, inp):
        if inp == Value.B1:
            self.z = 1
            return 1
        if self.z % 2 == 0:
            if inp == Value.A1 or inp == Value.A3:
                self.z += 1
            return 0
        if self.z % 2 == 1:
            self.z = 0
```

```
            self.state = States.P
            return 1

    def _reset(self):
        assert isinstance(self, System)
        self.z = System.__VALUE_INITIAL
        self.state = System.__STATES_INITIAL

    def process(self, seq):
        self._reset()
        assert self.z == System.__VALUE_INITIAL
        assert self.state == System.__STATES_INITIAL
        output = []
        for i in seq:
            o = self._input(i)
            output.append(o)
        return output
```

test_avtomat.py

```
import unittest
import avtomat
from avtomat import Value

W = [
    [Value.B1, Value.A0, Value.A0],
    [Value.B1, Value.A3],
    [Value.A0, Value.B1, Value.A0],
    [Value.A0, Value.A0, Value.A0, Value.A0],
    [Value.A0, Value.A0, Value.A3],
    [Value.A0, Value.A1, Value.A0, Value.A0],
    [Value.A0, Value.A1, Value.A3],
    [Value.A0, Value.A2, Value.A0, Value.A0],
    [Value.A0, Value.A2, Value.A3],
    [Value.A0, Value.A3, Value.A0, Value.A0],
    [Value.A0, Value.A3, Value.A3],
    [Value.A1, Value.A0],
    [Value.A2, Value.A0],
    [Value.A3, Value.B1, Value.A0, Value.A0],
    [Value.A3, Value.B1, Value.A3],
    [Value.A3, Value.A0, Value.B1, Value.A0],
    [Value.A3, Value.A0, Value.A0, Value.A0],
    [Value.A3, Value.A0, Value.A0, Value.A3],
    [Value.A3, Value.A0, Value.A1, Value.A0],
    [Value.A3, Value.A0, Value.A2, Value.A0],
    [Value.A3, Value.A0, Value.A2, Value.A3],
    [Value.A3, Value.A0, Value.A3, Value.A0],
    [Value.A3, Value.A1, Value.A0],
    [Value.A3, Value.A1, Value.A3],
    [Value.A3, Value.A2, Value.A0],
    [Value.A3, Value.A2, Value.A3],
    [Value.A3, Value.A3, Value.A0, Value.A0],
```

```
    [Value.A3, Value.A3, Value.A3]
]
out_W = [
    [1, 0, 1],
    [1, 1],
    [0, 1, 1],
    [0, 1, 0, 1],
    [0, 1, 1],
    [0, 1, 0, 1],
    [0, 1, 1],
    [0, 1, 0, 1],
    [0, 1, 1],
    [0, 1, 0, 1],
    [0, 1, 1],
    [0, 1],
    [0, 1],
    [1, 1, 0, 1],
    [1, 1, 1],
    [1, 0, 1, 1],
    [1, 0, 0, 0],
    [1, 0, 0, 0],
    [1, 0, 0, 1],
    [1, 0, 0, 0],
    [1, 0, 0, 0],
    [1, 0, 0, 1],
    [1, 0, 0],
    [1, 0, 0],
    [1, 0, 0],
    [1, 0, 0],
    [1, 1, 0, 1],
    [1, 1, 1]
]


class TestStringMethods(unittest.TestCase):
    def test_hsi(self):
        ins = avtomat.System()
        for (_input, _output) in list(zip(W, out_W)):
            self.assertEqual(ins.process(_input), _output)


if __name__ == '__main__':
    unittest.main()
```

Приложение 2. Текстовое описание автомата

```
s 4 S10 S11 S21 S22
i 5 A0 A1 A2 A3 B1
o 2 0 1
n0 S10
p 20
S10 B1 S10 1
S10 A0 S21 0
S10 A1 S21 0
S10 A2 S21 0
S10 A3 S11 1
S11 B1 S10 1
S11 A0 S22 0
S11 A1 S22 0
S11 A2 S22 0
S11 A3 S10 1
S21 B1 S21 1
S21 A0 S10 1
S21 A1 S10 1
S21 A2 S10 1
S21 A3 S10 1
S22 B1 S21 1
S22 A0 S22 0
S22 A1 S21 0
S22 A2 S22 0
S22 A3 S21 0
```