НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»
Факультет компьютерных наук
Базовая кафедра ИСП РАН

Лабораторная работа №1

«Построение тестов W-методом для системы, описанной в виде конечного автомата»

Выполнил:
студент группы мСП21
Казьмин Сергей

Москва
2021

1. Введение

Лабораторная работа предполагает несколько этапов:

1. выбор системы для тестирования;
2. построение спецификации выбранной системы в виде конечного автомата;
3. построение проверяющих тестов W-методом;
4. разработка программной реализации системы;
5. проведение тестирования программной реализации на сгенерированных тестовых последовательностях;
6. оценка полноты тесты относительно множества синтезированных мутантов.

2. Описание тестируемой системы

В качестве тестируемой системы выбрана дверь с ключ-картой.

При помощи ключ-карты можно открыть и закрыть замок, причём при открытии замок издаёт звуковой сигнал 1, а при закрытии звуковой сигнал 2.

Если замок разблокирован, то дверь можно свободно открывать и закрывать.

Если достать ключ-карту, когда дверь открыта, то замок заблокируется и дверь можно будет закрыть, после чего открыть её можно только разблокировав замок.

Если дверь закрыта, а замок заблокирован, то дверь можно открыть, сломав замок. В таком случае при открытии двери сработает тревога. Когда замок сломан, дверь можно свободно открывать и закрывать, а при использовании ключ-карты не будет происходить изменений в состоянии замка, так как он сломан.

3. Автоматное описание системы

На основе неформальной спецификации системы была построена формальная автоматная спецификация, представленная на рисунке 1.

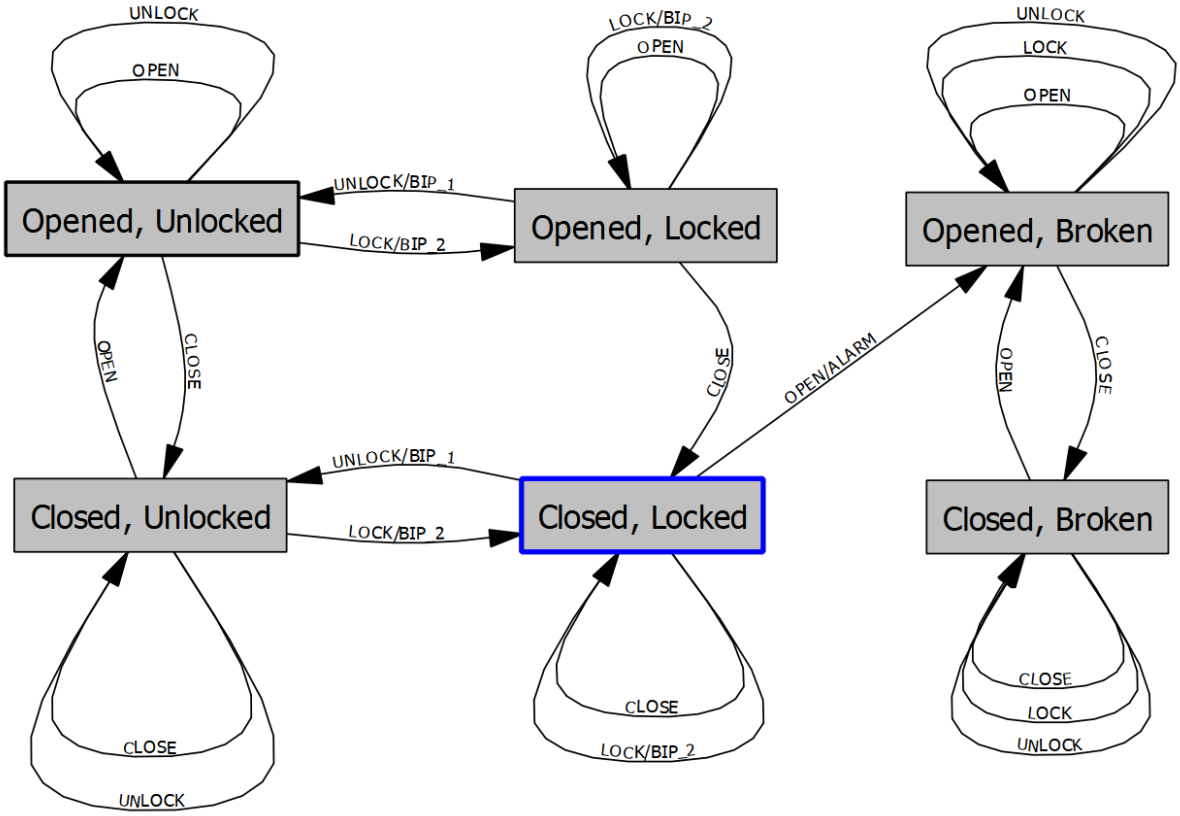

Рисунок 1 - Граф конечного автомата, описывающего дверь с автоматическим замком

Всего у автомата 6 состояний:

- Opened,Unlocked (дверь открыта, замок разблокирован);
- Opened,Locked (дверь открыта, замок заблокирован);
- Closed,Unlocked (дверь закрыта, замок разблокирован);
- Closed,Locked (дверь закрыта, замок заблокирован);
- Opened,Broken (дверь открыта, замок сломан);
- Closed,Broken (дверь закрыта, замок сломан).

4 входных символа:

- OPEN (открыть дверь);
- CLOSE (закрыть дверь);
- LOCK (разблокировать замок ключ-картой);
- UNLOCK (заблокировать замок, достав ключ-карту).

3 выходных символа:

- ALARM (срабатывание тревоги);
- BIP_1 (звуковой сигнал при разблокировке замка);
- BIP_2 (звуковой сигнал при блокировке замка).

В случае, когда выходной символ для перехода не указан, считается, что на выход подаётся пустой символ или же NULL.

Синей рамкой помечено начальное состояние автомата.

В представленном виде автомат не является минимальным, его можно минимизировать, склеив два состояния Opened,Broken и Closed,Broken в одно состояние - Broken. Полученный минимальный автомат представлен на рисунке 2.

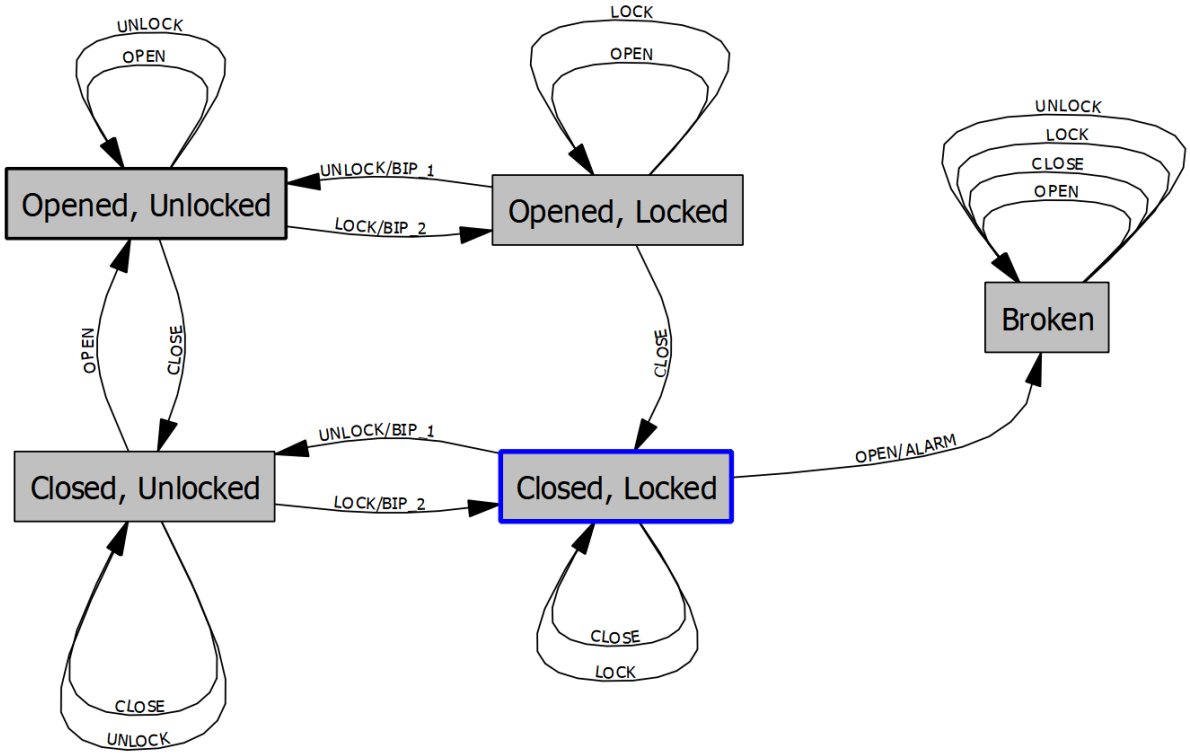

Рисунок 2 - Граф минимального конечного автомата, описывающего дверь с автоматическим замком

На основе построенной графовой схемы построено текстовая спецификация автомата в формате FSM. Представлено в приложении А.

С помощью портала fsmtestonline были построены тесты для загруженной текстовой спецификации в формате FSM. В качестве метода построения тестов был выбран метод “black box/W”.

Всего было сгенерировано 48 тестовых последовательностей, максимальная длина последовательностей равна 6. Сгенерированные последовательности представлены в приложении Б.

4. Описание программной реализации и процедуры тестирования

Программная реализация была выполнена на языке программирования Python и представлена в приложении В. Реализован класс AutomaticDoor, который имитирует поведение описанной ранее системы.

Для тестирования был использован библиотечный модуль unittest. С помощью реализованной функции test_fsm тестовые последовательности подавались на вход реализации, а выходные символы считывались и запомнилась. Далее с помощью модуля unittest был сформирован набор тестов, которые автоматически запускались перед началом этапа генерации мутантов.

5. Описание процедуры генерации мутантов

Для генерации мутантов был использован библиотечный модуль языка Python mutpy.

Генерация происходит с помощью модификации абстрактного синтаксического дерева исходной программы и поддерживает следующие мутации:

- удаление и перестановка арифметических операторов;
- перестановка операторов присваивания;
- удаление условных операторов;
- замена констант;
- другие.

Запуск генерации производился из консоли с помощью следующей команды:

```
mut.py --target automatic_door_fsm --unit-test main -e -m
```

В начале консольная утилита запускала все тесты из набора в файле main.py на реализации системы без мутаций из файла automatic_door_fsm.py, после чего запускает мутации.

Каждый мутант тестируется на сформированных тестах. После выполнения всех тестов для всех мутантов выводится статистика прохождения тестов.

6. Результаты мутационного тестирования

Консольная утилита для мутационного тестирования завершилась со следующим результатом.

[*] Start mutation process:

- targets: automatic_door_fsm
- tests: main

[*] 48 tests passed:

- main [0.00500 s]

[*] Start mutants generation and execution: ...

[*] Mutation score [1.20546 s]: 97.6%

- all: 146
- killed: 120 (82.2%)
- survived: 3 (2.1%)
- incompetent: 23 (15.8%)
- timeout: 0 (0.0%)

Все тесты были успешно пройдены для исходной реализации системы.

Всего было сгенерировано 146 мутантов.

Из них было обнаружено 120 (82.2%).

Не было обнаружено 3 мутанта (2.1%).

Оказались невалидными для выполнения интерпретатором Python 23 мутанта (15.8%).

Таким образом, полнота тестов относительно множества построенных мутантов составила 82.2%. Если исключить из набора невалидных мутантов, то полнота составит 97.6%.

Приложение А

Спецификация системы в формате FSM

```
F 0
s 5 Opened,Unlocked Opened,Locked Closed,Unlocked Closed,Locked Broken
i 4 open close lock unlock
o 4 ALARM BIP_1 BIP_2 NULL
n0 Closed,Locked
p 20
Opened,Unlocked open Opened,Unlocked NULL
Opened,Unlocked close Closed,Unlocked NULL
Opened,Unlocked lock Opened,Locked BIP_2
Opened,Unlocked unlock Opened,Unlocked NULL
Opened,Locked open Opened,Locked NULL
Opened,Locked close Closed,Locked NULL
Opened,Locked lock Opened,Locked NULL
Opened,Locked unlock Opened,Unlocked BIP_1
Closed,Unlocked open Opened,Unlocked NULL
Closed,Unlocked close Closed,Unlocked NULL
Closed,Unlocked lock Closed,Locked BIP_2
Closed,Unlocked unlock Closed,Unlocked NULL
Closed,Locked open Broken ALARM
Closed,Locked close Closed,Locked NULL
Closed,Locked lock Closed,Locked NULL
Closed,Locked unlock Closed,Unlocked BIP_1
Broken open Broken NULL
Broken close Broken NULL
Broken lock Broken NULL
Broken unlock Broken NULL
```

Приложение Б

Тестовые последовательности

open/ALARM open/NULL open/NULL

open/ALARM open/NULL lock/NULL open/NULL

open/ALARM open/NULL unlock/NULL

open/ALARM close/NULL open/NULL

open/ALARM close/NULL lock/NULL open/NULL

open/ALARM close/NULL unlock/NULL

open/ALARM lock/NULL open/NULL

open/ALARM lock/NULL lock/NULL open/NULL

open/ALARM lock/NULL unlock/NULL

open/ALARM unlock/NULL open/NULL

open/ALARM unlock/NULL lock/NULL open/NULL

open/ALARM unlock/NULL unlock/NULL

close/NULL open/ALARM

close/NULL lock/NULL open/ALARM

close/NULL unlock/BIP_1

lock/NULL open/ALARM

lock/NULL lock/NULL open/ALARM

lock/NULL unlock/BIP_1

unlock/BIP_1 open/NULL open/NULL open/NULL

unlock/BIP_1 open/NULL open/NULL lock/BIP_2 open/NULL

unlock/BIP_1 open/NULL open/NULL unlock/NULL

unlock/BIP_1 open/NULL close/NULL open/NULL

unlock/BIP_1 open/NULL close/NULL lock/BIP_2 open/ALARM

unlock/BIP_1 open/NULL close/NULL unlock/NULL

unlock/BIP_1 open/NULL lock/BIP_2 open/NULL open/NULL

unlock/BIP_1 open/NULL lock/BIP_2 open/NULL lock/NULL open/NULL

unlock/BIP_1 open/NULL lock/BIP_2 open/NULL unlock/BIP_1

unlock/BIP_1 open/NULL lock/BIP_2 close/NULL open/ALARM

unlock/BIP_1 open/NULL lock/BIP_2 close/NULL lock/NULL open/ALARM

unlock/BIP_1 open/NULL lock/BIP_2 close/NULL unlock/BIP_1

unlock/BIP_1 open/NULL lock/BIP_2 lock/NULL open/NULL

unlock/BIP_1 open/NULL lock/BIP_2 lock/NULL lock/NULL open/NULL

unlock/BIP_1 open/NULL lock/BIP_2 lock/NULL unlock/BIP_1

unlock/BIP_1 open/NULL lock/BIP_2 unlock/BIP_1 open/NULL

unlock/BIP_1 open/NULL lock/BIP_2 unlock/BIP_1 lock/BIP_2 open/NULL

unlock/BIP_1 open/NULL lock/BIP_2 unlock/BIP_1 unlock/NULL

unlock/BIP_1 open/NULL unlock/NULL open/NULL

unlock/BIP_1 open/NULL unlock/NULL lock/BIP_2 open/NULL

unlock/BIP_1 open/NULL unlock/NULL unlock/NULL

unlock/BIP_1 close/NULL open/NULL

unlock/BIP_1 close/NULL lock/BIP_2 open/ALARM

unlock/BIP_1 close/NULL unlock/NULL

unlock/BIP_1 lock/BIP_2 open/ALARM

unlock/BIP_1 lock/BIP_2 lock/NULL open/ALARM

unlock/BIP_1 lock/BIP_2 unlock/BIP_1

unlock/BIP_1 unlock/NULL open/NULL

unlock/BIP_1 unlock/NULL lock/BIP_2 open/ALARM

unlock/BIP_1 unlock/NULL unlock/NULL

Приложение В

Исходный код реализации и тестов

automatic_door_fsm.py

```
class AutomaticDoor:

    def __init__(self):
        self.state = 'Closed,Locked'
        self.out = ''

    def open(self):
        if self.state == 'Opened,Unlocked':
            self.null()
        elif self.state == 'Opened,Locked':
            self.null()
        elif self.state == 'Closed,Unlocked':
            self.state = 'Opened,Unlocked'
            self.null()
        elif self.state == 'Closed,Locked':
            self.state = 'Broken'
            self.alarm()
        elif self.state == 'Broken':
            self.null()

    def close(self):
        if self.state == 'Opened,Unlocked':
            self.state = 'Closed,Unlocked'
            self.null()
        elif self.state == 'Opened,Locked':
            self.state = 'Closed,Locked'
            self.null()
```

```
    elif self.state == 'Closed,Unlocked':
        self.null()
    elif self.state == 'Closed,Locked':
        self.null()
    elif self.state == 'Broken':
        self.null()

def lock(self):
    if self.state == 'Opened,Unlocked':
        self.state = 'Opened,Locked'
        self.bip_2()
    elif self.state == 'Opened,Locked':
        self.null()
    elif self.state == 'Closed,Unlocked':
        self.state = 'Closed,Locked'
        self.bip_2()
    elif self.state == 'Closed,Locked':
        self.null()
    elif self.state == 'Broken':
        self.null()

def unlock(self):
    if self.state == 'Opened,Unlocked':
        self.null()
    elif self.state == 'Opened,Locked':
        self.state = 'Opened,Unlocked'
        self.bip_1()
    elif self.state == 'Closed,Unlocked':
        self.null()
    elif self.state == 'Closed,Locked':
        self.state = 'Closed,Unlocked'
        self.bip_1()
```

```
        elif self.state == 'Broken':
            self.null()

    def null(self):
        self.out = 'NULL'

    def alarm(self):
        self.out = 'ALARM'

    def bip_1(self):
        self.out = 'BIP_1'

    def bip_2(self):
        self.out = 'BIP_2'
```

main.py

```
from unittest import TestCase
from automatic_door_fsm import AutomaticDoor


def test_fsm(input_seq):
    door = AutomaticDoor()
    out = str()
    transits = input_seq.split(' ')
    for t in transits:
        trigger = t.split('/')[0]
        eval(''.join(['door.', trigger, '()']))
        out += door.out + ' '
    return out[:-1]
```

```
class FSMTest(TestCase):

    def test_0(self):

        self.assertEqual(test_fsm('open/ALARM open/NULL open/NULL'),
'ALARM NULL NULL')


    def test_1(self):

        self.assertEqual(test_fsm('open/ALARM open/NULL lock/NULL
open/NULL'), 'ALARM NULL NULL NULL')


    def test_2(self):

        self.assertEqual(test_fsm('open/ALARM open/NULL unlock/NULL'),
'ALARM NULL NULL')


    def test_3(self):

        self.assertEqual(test_fsm('open/ALARM close/NULL open/NULL'),
'ALARM NULL NULL')


    def test_4(self):

        self.assertEqual(test_fsm('open/ALARM close/NULL lock/NULL
open/NULL'), 'ALARM NULL NULL NULL')


    def test_5(self):

        self.assertEqual(test_fsm('open/ALARM close/NULL unlock/NULL'),
'ALARM NULL NULL')


    def test_6(self):

        self.assertEqual(test_fsm('open/ALARM lock/NULL open/NULL'),
'ALARM NULL NULL')


    def test_7(self):

        self.assertEqual(test_fsm('open/ALARM lock/NULL lock/NULL
open/NULL'), 'ALARM NULL NULL NULL')


    def test_8(self):
```

```
        self.assertEqual(test_fsm('open/ALARM lock/NULL unlock/NULL'),
'ALARM NULL NULL')


    def test_9(self):

        self.assertEqual(test_fsm('open/ALARM unlock/NULL open/NULL'),
'ALARM NULL NULL')


    def test_10(self):

        self.assertEqual(test_fsm('open/ALARM unlock/NULL lock/NULL
open/NULL'), 'ALARM NULL NULL NULL')


    def test_11(self):

        self.assertEqual(test_fsm('open/ALARM unlock/NULL unlock/NULL'),
'ALARM NULL NULL')


    def test_12(self):

        self.assertEqual(test_fsm('close/NULL open/ALARM'), 'NULL ALARM')


    def test_13(self):

        self.assertEqual(test_fsm('close/NULL lock/NULL open/ALARM'),
'NULL NULL ALARM')


    def test_14(self):

        self.assertEqual(test_fsm('close/NULL unlock/BIP_1'), 'NULL
BIP_1')


    def test_15(self):

        self.assertEqual(test_fsm('lock/NULL open/ALARM'), 'NULL ALARM')


    def test_16(self):

        self.assertEqual(test_fsm('lock/NULL lock/NULL open/ALARM'), 'NULL
NULL ALARM')


    def test_17(self):
```

```
        self.assertEqual(test_fsm('lock/NULL unlock/BIP_1'), 'NULL BIP_1')


    def test_18(self):

        self.assertEqual(test_fsm('unlock/BIP_1 open/NULL open/NULL
open/NULL'), 'BIP_1 NULL NULL NULL')


    def test_19(self):

        self.assertEqual(test_fsm('unlock/BIP_1 open/NULL open/NULL
lock/BIP_2 open/NULL'),

                         'BIP_1 NULL NULL BIP_2 NULL')


    def test_20(self):

        self.assertEqual(test_fsm('unlock/BIP_1 open/NULL open/NULL
unlock/NULL'), 'BIP_1 NULL NULL NULL')


    def test_21(self):

        self.assertEqual(test_fsm('unlock/BIP_1 open/NULL close/NULL
open/NULL'), 'BIP_1 NULL NULL NULL')


    def test_22(self):

        self.assertEqual(test_fsm('unlock/BIP_1 open/NULL close/NULL
lock/BIP_2 open/ALARM'),

                         'BIP_1 NULL NULL BIP_2 ALARM')


    def test_23(self):

        self.assertEqual(test_fsm('unlock/BIP_1 open/NULL close/NULL
unlock/NULL'), 'BIP_1 NULL NULL NULL')


    def test_24(self):

        self.assertEqual(test_fsm('unlock/BIP_1 open/NULL lock/BIP_2
open/NULL open/NULL'),

                         'BIP_1 NULL BIP_2 NULL NULL')


    def test_25(self):
```

```
        self.assertEqual(test_fsm('unlock/BIP_1 open/NULL lock/BIP_2
open/NULL lock/NULL open/NULL'),
                         'BIP_1 NULL BIP_2 NULL NULL NULL')


    def test_26(self):
        self.assertEqual(test_fsm('unlock/BIP_1 open/NULL lock/BIP_2
open/NULL unlock/BIP_1'),
                         'BIP_1 NULL BIP_2 NULL BIP_1')


    def test_27(self):
        self.assertEqual(test_fsm('unlock/BIP_1 open/NULL lock/BIP_2
close/NULL open/ALARM'),
                         'BIP_1 NULL BIP_2 NULL ALARM')


    def test_28(self):
        self.assertEqual(test_fsm('unlock/BIP_1 open/NULL lock/BIP_2
close/NULL lock/NULL open/ALARM'),
                         'BIP_1 NULL BIP_2 NULL NULL ALARM')


    def test_29(self):
        self.assertEqual(test_fsm('unlock/BIP_1 open/NULL lock/BIP_2
close/NULL unlock/BIP_1'),
                         'BIP_1 NULL BIP_2 NULL BIP_1')


    def test_30(self):
        self.assertEqual(test_fsm('unlock/BIP_1 open/NULL lock/BIP_2
lock/NULL open/NULL'),
                         'BIP_1 NULL BIP_2 NULL NULL')


    def test_31(self):
        self.assertEqual(test_fsm('unlock/BIP_1 open/NULL lock/BIP_2
lock/NULL lock/NULL open/NULL'),
                         'BIP_1 NULL BIP_2 NULL NULL NULL')
```

```
    def test_32(self):

        self.assertEqual(test_fsm('unlock/BIP_1 open/NULL lock/BIP_2
lock/NULL unlock/BIP_1'),

                         'BIP_1 NULL BIP_2 NULL BIP_1')


    def test_33(self):

        self.assertEqual(test_fsm('unlock/BIP_1 open/NULL lock/BIP_2
unlock/BIP_1 open/NULL'),

                         'BIP_1 NULL BIP_2 BIP_1 NULL')


    def test_34(self):

        self.assertEqual(test_fsm('unlock/BIP_1 open/NULL lock/BIP_2
unlock/BIP_1 lock/BIP_2 open/NULL'),

                         'BIP_1 NULL BIP_2 BIP_1 BIP_2 NULL')


    def test_35(self):

        self.assertEqual(test_fsm('unlock/BIP_1 open/NULL lock/BIP_2
unlock/BIP_1 unlock/NULL'),

                         'BIP_1 NULL BIP_2 BIP_1 NULL')


    def test_36(self):

        self.assertEqual(test_fsm('unlock/BIP_1 open/NULL unlock/NULL
open/NULL'), 'BIP_1 NULL NULL NULL')


    def test_37(self):

        self.assertEqual(test_fsm('unlock/BIP_1 open/NULL unlock/NULL
lock/BIP_2 open/NULL'),

                         'BIP_1 NULL NULL BIP_2 NULL')


    def test_38(self):

        self.assertEqual(test_fsm('unlock/BIP_1 open/NULL unlock/NULL
unlock/NULL'), 'BIP_1 NULL NULL NULL')


    def test_39(self):
```

```
        self.assertEqual(test_fsm('unlock/BIP_1 close/NULL open/NULL'),
'BIP_1 NULL NULL')


    def test_40(self):

        self.assertEqual(test_fsm('unlock/BIP_1 close/NULL lock/BIP_2
open/ALARM'), 'BIP_1 NULL BIP_2 ALARM')


    def test_41(self):

        self.assertEqual(test_fsm('unlock/BIP_1 close/NULL unlock/NULL'),
'BIP_1 NULL NULL')


    def test_42(self):

        self.assertEqual(test_fsm('unlock/BIP_1 lock/BIP_2 open/ALARM'),
'BIP_1 BIP_2 ALARM')


    def test_43(self):

        self.assertEqual(test_fsm('unlock/BIP_1 lock/BIP_2 lock/NULL
open/ALARM'), 'BIP_1 BIP_2 NULL ALARM')


    def test_44(self):

        self.assertEqual(test_fsm('unlock/BIP_1 lock/BIP_2 unlock/BIP_1'),
'BIP_1 BIP_2 BIP_1')


    def test_45(self):

        self.assertEqual(test_fsm('unlock/BIP_1 unlock/NULL open/NULL'),
'BIP_1 NULL NULL')


    def test_46(self):

        self.assertEqual(test_fsm('unlock/BIP_1 unlock/NULL lock/BIP_2
open/ALARM'), 'BIP_1 NULL BIP_2 ALARM')


    def test_47(self):

        self.assertEqual(test_fsm('unlock/BIP_1 unlock/NULL unlock/NULL'),
'BIP_1 NULL NULL')
```