



Национальный исследовательский университет «Высшая школа экономики»  
Факультет компьютерных наук  
Системное программирование

Оплачко Николай Алексеевич  
МСП21

## **Лабораторная работа 1**

Формальные методы программной инженерии

# Оглавление

1	Введение . . . . .	3
2	Неформальное описание выбранной системы . . . . .	3
3	Описание программной реализации . . . . .	4
4	Автоматное описание системы, использованный метод построения тестовых последовательностей и построенное множество тестовых последовательностей . . . . .	5
5	Процедура генерации и описание построенного множества мутантов . . . . .	9
6	Описание процедуры тестирования программной реализации и найденные ошибки . . . . .	10
7	Оценка полноты теста относительно множества построенных мутантов . . . . .	14
8	Заключение . . . . .	14
1	Приложение: исходный код . . . . .	15

# 1. Введение

Цели данной работы:

- Предложить систему и привести её неформальное описание
- Привести формальную спецификацию системы в виде конечного автомата
- Построить тестовые последовательности для автомата
- Программно реализовать предложенную систему и сгенерировать множество её мутантов
- Оценить полноту набора тестовых последовательностей относительно множества мутантов

## 2. Неформальное описание выбранной системы

Задача системы — посчитать количество чисел, поданных на вход, кратных трём.

1) На вход системе подаётся текст, состоящий из цифр и пробелов:

- непрерывные группы цифр считаются числом, записанным в десятичной системе счисления.

2) Система считывает текст посимвольно, считая пробел отдельным символом.

3) При обнаружении числа, кратного трём, система возвращает команду «+1».

Примеры:

- $\square\square \rightarrow 0$  чисел кратно трём
- $1\square2\square \rightarrow 0$  чисел кратно трём
- $4\square\square33\square \rightarrow 1$  число кратно трём

### 3. Описание программной реализации

В приложении 1 приведён исходный код программной реализации на языке C.

В файле `src/div.c` программно реализована процедура `scan_numbers_divisible_by_three`, которая принимает в качестве аргумента строку с трансформированным тестовым входом и выводит на каждый символ реакцию: 0 или 1 (аналогично автомату). Суть происходящего — к считываемому числу приписывается по одной цифре и пересчитывается остаток по модулю три. Если число считано целиком (встречен пробел), и остаток оказался равен нулю, программа выводит 1. В остальных случаях реакцией на символ будет 0.

Точка входа программной реализации находится в файле `src/main.c`: считывается строка и подаётся в качестве аргумента в процедуру `scan_numbers_divisible_by_three`.

Процесс сборки программной реализации описан в `CMakeLists.txt` и иницируется для заданного мутанта запуском скрипта `scripts/build_with_mutant.sh`.

Скрипт `scripts/transform_input.sh` осуществляет трансформацию входной тестовой последовательности. Пример: `123_45_ → 1203123`.

Скрипт `scripts/generate_mutants.sh` генерирует набор мутантов и оставляет только успешно проходящих процесс сборки.

Скрипт `scripts/run_tests.py` запускает исполняемый файл на заданном наборе тестовых последовательностей и завершается с одним из исходов:

- Тесты успешно пройдены
- Получен неправильный ответ на один из тестов
- Программа завершилась с ошибкой
- Программа превысила временное ограничение

Скрипт `scripts/test_mutants.sh` иницирует для каждого мутанта скрипт сборки, затем скрипт тестирования на заданном наборе тестовых последовательностей и сохраняет набор мутантов, успешно прошедших все тесты.

#### 4. Автоматное описание системы, использованный метод построения тестовых последовательностей и построенное множество тестовых последовательностей

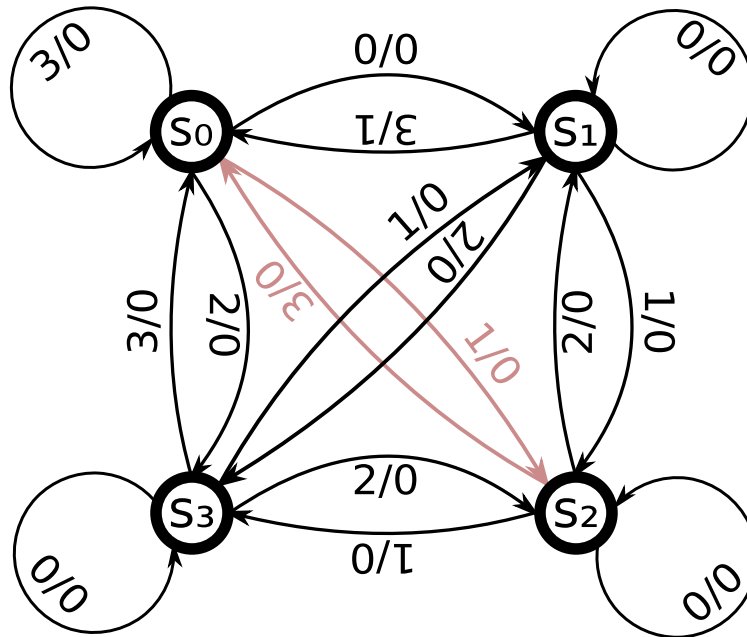


Рис. 1: Автоматное описание системы. Начальное состояние  $s_0$

С целью упрощения формализации в виде автомата предложенной системы, её вход преобразуется следующей подстановкой:

$$0 \mid 3 \mid 6 \mid 9 \rightarrow 0$$

$$1 \mid 4 \mid 7 \rightarrow 1$$

$$2 \mid 5 \mid 8 \rightarrow 2$$

$$\langle \text{space} \rangle \rightarrow 3$$

Суть этого преобразования в том, что мы считаем равнозначными цифры, дающие одинаковый остаток при делении на три. На рис. 1 графически изображен автомат, формально описывающий предложенную систему. Листинг 1 содержит описание этого автомата в формате FSM. Его реакцией на каждый допустимый входной символ являются 1, если было обнаружено очередное число, кратное трём, и 0 иначе. Данный автомат по сути считает сумму цифр в числе по модулю три, переходя в соответствующее значению остатка состояние. Корректность преобразования и работы автомата следует

из свойств взятия модуля от арифметических выражений:

$$\begin{aligned} & (a_0 + a_1 \cdot 10 + \dots + a_n \cdot 10^n) \% 3 = \\ & ((a_0 \% 3) + (a_1 \% 3) \cdot 10 + \dots + (a_n \% 3) \cdot 10^n) \% 3 = \\ & ((a_0 \% 3) + (a_1 \% 3) + \dots + (a_n \% 3)) \% 3 \end{aligned}$$

Для генерации множества тестовых последовательностей был использован метод «Black box/W» с заданными максимальными размерами конечного автомата 4 и 5, реализованный в системе FSM Tester<sup>1</sup>. В листингах 2 и 3 приведены сгенерированные тестовые последовательности.

#### Листинг 1: FSM-описание автомата

```

1  F 0
2  s 4
3  i 4
4  o 2
5  n0 0
6  p 16
7  0 0 1 0
8  0 1 2 0
9  0 2 3 0
10 0 3 0 0
11 1 0 1 0
12 1 1 2 0
13 1 2 3 0
14 1 3 0 1
15 2 0 2 0
16 2 1 3 0
17 2 2 1 0
18 2 3 0 0
19 3 0 3 0
20 3 1 1 0
21 3 2 2 0
22 3 3 0 0

```

---

<sup>1</sup><http://www.fsmtestonline.ru>

## Листинг 2: Тестовые последовательности для размера 4

```
1  0/0 0/0 0/0 3/1
2  0/0 0/0 1/0 3/0
3  0/0 0/0 3/1
4  0/0 1/0 0/0 3/0
5  0/0 1/0 1/0 3/0
6  0/0 1/0 3/0
7  0/0 2/0 0/0 3/0
8  0/0 2/0 1/0 3/1
9  0/0 2/0 3/0
10 0/0 3/1 0/0 3/1
11 0/0 3/1 1/0 3/0
12 0/0 3/1 3/0
13 1/0 0/0 0/0 3/0
14 1/0 0/0 1/0 3/0
15 1/0 0/0 3/0
16 1/0 1/0 0/0 3/0
17 1/0 1/0 1/0 3/1
18 1/0 1/0 3/0
19 1/0 2/0 0/0 3/1
20 1/0 2/0 1/0 3/0
21 1/0 2/0 3/1
22 1/0 3/0 0/0 3/1
23 1/0 3/0 1/0 3/0
24 1/0 3/0 3/0
25 2/0 0/0 0/0 3/0
26 2/0 0/0 1/0 3/1
27 2/0 0/0 3/0
28 2/0 1/0 0/0 3/1
29 2/0 1/0 1/0 3/0
30 2/0 1/0 3/1
31 2/0 2/0 0/0 3/0
32 2/0 2/0 1/0 3/0
33 2/0 2/0 3/0
34 2/0 3/0 0/0 3/1
35 2/0 3/0 1/0 3/0
36 2/0 3/0 3/0
37 3/0 0/0 3/1
38 3/0 1/0 3/0
39 3/0 3/0
```

## Листинг 3: Тестовые последовательности для размера 5

1	0/0	0/0	0/0	0/0	3/1	79	1/0	2/0	2/0	0/0	3/0
2	0/0	0/0	0/0	1/0	3/0	80	1/0	2/0	2/0	1/0	3/1
3	0/0	0/0	0/0	3/1		81	1/0	2/0	2/0	3/0	
4	0/0	0/0	1/0	0/0	3/0	82	1/0	2/0	3/1	0/0	3/1
5	0/0	0/0	1/0	1/0	3/0	83	1/0	2/0	3/1	1/0	3/0
6	0/0	0/0	1/0	3/0		84	1/0	2/0	3/1	3/0	
7	0/0	0/0	2/0	0/0	3/0	85	1/0	3/0	0/0	0/0	3/1
8	0/0	0/0	2/0	1/0	3/1	86	1/0	3/0	0/0	1/0	3/0
9	0/0	0/0	2/0	3/0		87	1/0	3/0	0/0	3/1	
10	0/0	0/0	3/1	0/0	3/1	88	1/0	3/0	1/0	0/0	3/0
11	0/0	0/0	3/1	1/0	3/0	89	1/0	3/0	1/0	1/0	3/0
12	0/0	0/0	3/1	3/0		90	1/0	3/0	1/0	3/0	
13	0/0	1/0	0/0	0/0	3/0	91	1/0	3/0	2/0	0/0	3/0
14	0/0	1/0	0/0	1/0	3/0	92	1/0	3/0	2/0	1/0	3/1
15	0/0	1/0	0/0	3/0		93	1/0	3/0	2/0	3/0	
16	0/0	1/0	1/0	0/0	3/0	94	1/0	3/0	3/0	0/0	3/1
17	0/0	1/0	1/0	1/0	3/1	95	1/0	3/0	3/0	1/0	3/0
18	0/0	1/0	1/0	3/0		96	1/0	3/0	3/0	3/0	
19	0/0	1/0	2/0	0/0	3/1	97	2/0	0/0	0/0	0/0	3/0
20	0/0	1/0	2/0	1/0	3/0	98	2/0	0/0	0/0	1/0	3/1
21	0/0	1/0	2/0	3/1		99	2/0	0/0	0/0	3/0	
22	0/0	1/0	3/0	0/0	3/1	100	2/0	0/0	1/0	0/0	3/1
23	0/0	1/0	3/0	1/0	3/0	101	2/0	0/0	1/0	1/0	3/0
24	0/0	1/0	3/0	3/0		102	2/0	0/0	1/0	3/1	
25	0/0	2/0	0/0	0/0	3/0	103	2/0	0/0	2/0	0/0	3/0
26	0/0	2/0	0/0	1/0	3/1	104	2/0	0/0	2/0	1/0	3/0
27	0/0	2/0	0/0	3/0		105	2/0	0/0	2/0	3/0	
28	0/0	2/0	1/0	0/0	3/1	106	2/0	0/0	3/0	0/0	3/1
29	0/0	2/0	1/0	1/0	3/0	107	2/0	0/0	3/0	1/0	3/0
30	0/0	2/0	1/0	3/1		108	2/0	0/0	3/0	3/0	
31	0/0	2/0	2/0	0/0	3/0	109	2/0	1/0	0/0	0/0	3/1
32	0/0	2/0	2/0	1/0	3/0	110	2/0	1/0	0/0	1/0	3/0
33	0/0	2/0	2/0	3/0		111	2/0	1/0	0/0	3/1	
34	0/0	2/0	3/0	0/0	3/1	112	2/0	1/0	1/0	0/0	3/0
35	0/0	2/0	3/0	1/0	3/0	113	2/0	1/0	1/0	1/0	3/0
36	0/0	2/0	3/0	3/0		114	2/0	1/0	1/0	3/0	
37	0/0	3/1	0/0	0/0	3/1	115	2/0	1/0	2/0	0/0	3/0
38	0/0	3/1	0/0	1/0	3/0	116	2/0	1/0	2/0	1/0	3/1
39	0/0	3/1	0/0	3/1		117	2/0	1/0	2/0	3/0	
40	0/0	3/1	1/0	0/0	3/0	118	2/0	1/0	3/1	0/0	3/1
41	0/0	3/1	1/0	1/0	3/0	119	2/0	1/0	3/1	1/0	3/0
42	0/0	3/1	1/0	3/0		120	2/0	1/0	3/1	3/0	
43	0/0	3/1	2/0	0/0	3/0	121	2/0	2/0	0/0	0/0	3/0
44	0/0	3/1	2/0	1/0	3/1	122	2/0	2/0	0/0	1/0	3/0
45	0/0	3/1	2/0	3/0		123	2/0	2/0	0/0	3/0	
46	0/0	3/1	3/0	0/0	3/1	124	2/0	2/0	1/0	0/0	3/0
47	0/0	3/1	3/0	1/0	3/0	125	2/0	2/0	1/0	1/0	3/1
48	0/0	3/1	3/0	3/0		126	2/0	2/0	1/0	3/0	
49	1/0	0/0	0/0	0/0	3/0	127	2/0	2/0	2/0	0/0	3/1
50	1/0	0/0	0/0	1/0	3/0	128	2/0	2/0	2/0	1/0	3/0
51	1/0	0/0	0/0	3/0		129	2/0	2/0	2/0	3/1	
52	1/0	0/0	1/0	0/0	3/0	130	2/0	2/0	3/0	0/0	3/1
53	1/0	0/0	1/0	1/0	3/1	131	2/0	2/0	3/0	1/0	3/0
54	1/0	0/0	1/0	3/0		132	2/0	2/0	3/0	3/0	
55	1/0	0/0	2/0	0/0	3/1	133	2/0	3/0	0/0	0/0	3/1
56	1/0	0/0	2/0	1/0	3/0	134	2/0	3/0	0/0	1/0	3/0
57	1/0	0/0	2/0	3/1		135	2/0	3/0	0/0	3/1	
58	1/0	0/0	3/0	0/0	3/1	136	2/0	3/0	1/0	0/0	3/0
59	1/0	0/0	3/0	1/0	3/0	137	2/0	3/0	1/0	1/0	3/0
60	1/0	0/0	3/0	3/0		138	2/0	3/0	1/0	3/0	
61	1/0	1/0	0/0	0/0	3/0	139	2/0	3/0	2/0	0/0	3/0
62	1/0	1/0	0/0	1/0	3/1	140	2/0	3/0	2/0	1/0	3/1
63	1/0	1/0	0/0	3/0		141	2/0	3/0	2/0	3/0	
64	1/0	1/0	1/0	0/0	3/1	142	2/0	3/0	3/0	0/0	3/1
65	1/0	1/0	1/0	1/0	3/0	143	2/0	3/0	3/0	1/0	3/0
66	1/0	1/0	1/0	3/1		144	2/0	3/0	3/0	3/0	
67	1/0	1/0	2/0	0/0	3/0	145	3/0	0/0	0/0	3/1	
68	1/0	1/0	2/0	1/0	3/0	146	3/0	0/0	1/0	3/0	
69	1/0	1/0	2/0	3/0		147	3/0	0/0	3/1		
70	1/0	1/0	3/0	0/0	3/1	148	3/0	1/0	0/0	3/0	
71	1/0	1/0	3/0	1/0	3/0	149	3/0	1/0	1/0	3/0	
72	1/0	1/0	3/0	3/0		150	3/0	1/0	3/0		
73	1/0	2/0	0/0	0/0	3/1	151	3/0	2/0	0/0	3/0	
74	1/0	2/0	0/0	1/0	3/0	152	3/0	2/0	1/0	3/1	
75	1/0	2/0	0/0	3/1		153	3/0	2/0	3/0		
76	1/0	2/0	1/0	0/0	3/0	154	3/0	3/0	0/0	3/1	
77	1/0	2/0	1/0	1/0	3/0	155	3/0	3/0	1/0	3/0	
78	1/0	2/0	1/0	3/0		156	3/0	3/0	3/0		



## 5. Процедура генерации и описание построенного множества мутантов

Для генерации множества мутантов был использован проект с открытым исходным кодом `universalmutator`<sup>2</sup>, основанный на регулярных выражениях. В листинге 4 приведён список правил для языка C, используемый мутатором для генерации программ-мутантов.

Листинг 4: Список правил-подстановок мутатора для языка C

```

1  (^\\s*)(\\S+[^{]}+.*\\n ==> \\1/*\\2*/\\n
2  if (\\(..*\\)) ==> if (!\\1)
3  if(\\(..*\\)) ==> if (!\\1)
4  if (\\(..*\\)) ==> if (0)
5  if(\\(..*\\)) ==> if(0)
6  if (\\(..*\\)) ==> if (1)
7  if(\\(..*\\)) ==> if(1)
8  while (\\(..*\\)) ==> while (!\\1)
9  while(\\(..*\\)) ==> while(!\\1)
10 else ==>
11 \\|\\|\\|..*\\) ==> || 0
12 \\|\\|\\|..*\\s ==> || 0
13 \\(..*\\|\\|\\| ==> 0 ||
14 \\s.*\\|\\|\\| ==> 0 ||
15 &&.*\\) ==> && 1
16 &&.*\\s ==> && 1
17 \\(..*&& ==> 1 &&
18 \\s.*&& ==> 1 &&
19 // ==> SKIP_MUTATING_REST

```

Генерация мутантов и фильтрация успешно компилируемых среди них производился запуском соответствующего скрипта:

```
$ ./scripts/generate_mutants.sh
```

---

<sup>2</sup>Исходный код `universalmutator`: <https://github.com/agroce/universalmutator>.

## 6. Описание процедуры тестирования программной реализации и найденные ошибки

Прогон сгенерированных мутантов на тестовых набор был произведён следующими командами:

```
$ ./scripts/test_mutants.sh ./tests/tests_4.txt
    passing_mutants_4.txt
$ ./scripts/test_mutants.sh ./tests/tests_5.txt
    passing_mutants_5.txt
```

Было получено 240 мутантов, которые мы разделим на следующие группы:

- 1) 56 мутантов проходят тесты для автомата размера 4
  - из них 53 мутанта также проходят тесты для автомата размера 5
    - 24 мутанта при компиляции дают оригинальный ассемблер
    - 23 мутанта семантически эквивалентны изначальной программной реализации при заданных ограничениях на вход
    - 6 мутантов эквивалентны изначальной программной реализации при заданных ограничениях на вход и на компилятор
- 2) 148 мутантов выводят неправильный ответ на один из тестов
- 3) 13 мутантов завершаются с ошибкой исполнения программы
- 4) 23 мутанта не выводят ответ на тест за установленное временное ограничение в 1 секунду

Три мутанта из первой группы сразу показывают неполноту множества тестовых последовательностей для автомата размера 4. Суть каждого из них — проигнорировать числа, состоящие целиком из цифр 2 (не установить флаг, означающий считывание числа). Минимальным контрпримером к программной реализации с такой ошибкой является тестовый пример 2/0 2/0 2/0 3/1, который отсутствует среди тестовых последовательностей для размера 4, но присутствует среди оных для размера 5.

## Подмножество мутантов, проходящих все сгенерированные тесты

Некоторые мутанты не являются эквивалентными программами на уровне сгенерированного кода ассемблера, однако продолжают работать за счёт соблюдения некоторых ограничений при выполнении программы. Например, в листинге 5 приведены примеры, работающие по причине того, что остаток от деления 10 на 3 равен 1.

Листинг 5: Мутанты, использующие модульную арифметику

```

1 mutants/div.mutant.7.c
2 6c6
3 < const int BASE = 10;
4 ---
5 > const int BASE = 1;
6 mutants/div.mutant.106.c
7 33c33
8 <             number_rem = (number_rem * BASE + 0) % MOD;
9 ---
10 >             number_rem = (number_rem % BASE + 0) % MOD;
```

Шесть мутантов среди проходящих все тесты используют переполнение (примеры в листинге 6). Программа продолжает корректно работать за счёт ограничений на входную последовательность, а также поскольку 0 и -1 для типа `unsigned` имеют одинаковый остаток по модулю 3:  $2^{32} - 1 \equiv 0 \pmod{3}$ . Переполнение не относится к `undefined behavior`, и при соответствии компилятора стандарту языка C на большинстве распространённых систем программа будет продолжать работать. Однако стандарт C гарантирует лишь то, что тип `unsigned` занимает как минимум 16 бит<sup>3</sup>, и если его ширина будет вида, отличного от  $2^n$ ,  $n \geq 4$ , то программа работать перестанет. Поскольку на используемой для тестирования операционной системе ширина типа `unsigned` равна 32, никакие тесты не в состоянии опровергнуть таких мутантов, и в итоговом зачёте мы их учитывать не будем.

Листинг 6: Мутанты, использующие переполнение типа `unsigned`

```

1 mutants/div.mutant.104.c
2 33c33
3 <             number_rem = (number_rem * BASE + 0) % MOD;
4 ---
5 >             number_rem = (number_rem - BASE + 0) % MOD;
6 mutants/div.mutant.76.c
```

<sup>3</sup>[https://en.cppreference.com/w/c/language/arithmetic\\_types](https://en.cppreference.com/w/c/language/arithmetic_types)

```

7 26c26
8 <      unsigned number_rem = 0;
9 ---
10 >      unsigned number_rem = -1;

```

## Подмножество мутантов, не укладывающихся во временное ограничение

В листинге 7 приведён пример мутанта, который вместо увеличения значения позиции в строке оставляет её неизменной, что приводит к заикливлению на одном символе.

Листинг 7: Мутант, вызывающий заикливление

```

1 mutants/div.mutant.43.c
2 18c18
3 < void emit(size_t* pos, char c) { ++*pos; putchar(c); }
4 ---
5 > void emit(size_t* pos, char c) { +-*pos; putchar(c); }

```

## Подмножество мутантов, завершающихся с ошибкой исполнения

Примеры мутантов, завершающихся с ошибкой, приведены в листинге 8: в первом примере мутатор переставил местами аргументы, подав на место указателя значение 0, и программа обратилась в память, в которую ей обращаться запретило ядро операционной системы; во втором примере мутатор заменил добавление нуля делением на него, и возникло исключение.

Листинг 8: Мутанты, завершающийся с ошибкой

```

1 mutants/div.mutant.97.c
2 32c32
3 <      emit(&i, OUT_ZERO);
4 ---
5 >      emit( OUT_ZERO,&i);
6 mutants/div.mutant.101.c
7 33c33
8 <      number_rem = (number_rem * BASE + 0) % MOD;
9 ---
10 >      number_rem = (number_rem * BASE / 0) % MOD;

```

## Подмножество мутантов, дающих ошибочный ответ на одну из тестовых последовательностей

Большая часть мутантов дают ошибочную реакцию на некоторые тестовые последовательности. Некоторые примеры таких мутантов приведены в листинге 9. Процесс их тестирования, содержащий тестовые последовательности, на которые мутанты дали неверные ответы, находится в листинге 10.

Листинг 9: Мутанты, выдающих ошибочную реакцию

```

1 mutants/div.mutant.38.c
2 16c16
3 < #define OUT_ONE '1'
4 ---
5 > #define OUT_ONE '0'
6 mutants/div.mutant.58.c
7 22c22
8 <     while (s[i] != IN_END) {
9 ---
10 >     if (s[i] != IN_END) {
11 mutants/div.mutant.182.c
12 43c43
13 <             number_rem = (number_rem * BASE + 2) % MOD;
14 ---
15 >             /*number_rem = (number_rem * BASE + 2) % MOD;*/
16 mutants/div.mutant.44.c
17 18c18
18 < void emit(size_t* pos, char c) { ++*pos; putchar(c); }
19 ---
20 > void emit(size_t* pos, char c) { --*pos; putchar(c); }
```

Листинг 10: Тестирование приведённых примеров ошибочных мутантов

```

1 $ ./scripts/build_with_mutant.sh mutants/div.mutant.38.c
2 $ python3 ./scripts/run_tests.py -v -e ./build/div tests/tests_4.txt
3 Failed on input: 0003
4 Program output: 0000
5 Expected output: 0001
6 $ ./scripts/build_with_mutant.sh mutants/div.mutant.58.c
7 $ python3 ./scripts/run_tests.py -v -e ./build/div tests/tests_4.txt
8 Failed on input: 0303
9 Program output: 01
10 Expected output: 0101
11 $ ./scripts/build_with_mutant.sh mutants/div.mutant.182.c
12 $ python3 ./scripts/run_tests.py -v -e ./build/div tests/tests_4.txt
13 Failed on input: 0203
14 Program output: 0001
15 Expected output: 0000
16 $ ./scripts/build_with_mutant.sh mutants/div.mutant.44.c
17 $ python3 ./scripts/run_tests.py -v -e ./build/div tests/tests_4.txt
18 Failed on input: 0003
19 Program output: 0
20 Expected output: 0001
```

## 7. Оценка полноты теста относительно множества построенных мутантов

Мутатором было построено множество из 240 мутантов, из которых 53 было отнесено к эквивалентным при некоторых описанных ограничениях. Среди оставшихся 187 мутантов — 3 мутанта проходят все сгенерированные тесты для автомата размера 4, но не проходят для размера 5. Таким образом:

- Полнота теста для размера 4 —  $\frac{187-3}{187} \cdot 100\% \approx 98.4\%$
- Полнота теста для размера 5 — 100%

## 8. Заключение

В настоящей работе произведено следующее:

- предложены неформальное и автоматное описание системы
- произведена программная реализация системы, исходный код которой находится на GitHub<sup>4</sup>
- построены два множества тестовых последовательностей методом «Black box/W» по автоматному описанию системы
- сгенерировано множество мутантов программной реализации
- полученные мутанты были протестированы и классифицированы
- было показано, что одно из множеств тестовых последовательностей не является полным относительно сгенерированного множества мутантов, другое же оказалось полным

---

<sup>4</sup><https://github.com/magnickolas/labwork-fsm-tests>

# 1. Приложение: исходный код

Листинг 11: scripts/transform\_input.sh

```

1  #!/bin/bash
2  tr 0 0 | tr 3 0 | tr 6 0 | tr 9 0 | \
3  tr 1 1 | tr 4 1 | tr 7 1 | \
4  tr 2 2 | tr 5 2 | tr 8 2 | \
5  tr ' ' 3

```

Листинг 12: src/div.c

```

1  #include "div.h"
2  #include <stdbool.h>
3  #include <stddef.h>
4  #include <stdio.h>
5
6  const int BASE = 10;
7  const int MOD = 3;
8
9  #define IN_ZERO '0'
10 #define IN_ONE '1'
11 #define IN_TWO '2'
12 #define IN_SPACE '3'
13 #define IN_END '\0'
14
15 #define OUT_ZERO '0'
16 #define OUT_ONE '1'
17
18 void emit(size_t* pos, char c) { ++*pos; putchar(c); }
19
20 void scan_numbers_divisible_by_three(char* s) {
21     size_t i = 0;
22     while (s[i] != IN_END) {
23         while (s[i] == IN_SPACE) {
24             emit(&i, OUT_ZERO);
25         }
26         unsigned number_rem = 0;
27         bool number_start = false;
28         bool number_end = false;
29         while (!number_end) {
30             switch (s[i]) {
31                 case IN_ZERO:
32                     emit(&i, OUT_ZERO);
33                     number_rem = (number_rem * BASE + 0) % MOD;
34                     number_start = true;
35                     break;
36                 case IN_ONE:
37                     emit(&i, OUT_ZERO);
38                     number_rem = (number_rem * BASE + 1) % MOD;
39                     number_start = true;
40                     break;
41                 case IN_TWO:
42                     emit(&i, OUT_ZERO);
43                     number_rem = (number_rem * BASE + 2) % MOD;
44                     number_start = true;
45                     break;
46                 default:

```

```

47         number_end = true;
48     }
49 }
50 if (s[i] != IN_END) {
51     if (number_start && number_rem == 0) {
52         emit(&i, OUT_ONE);
53     } else {
54         emit(&i, OUT_ZERO);
55     }
56 }
57 }
58 }

```

Листинг 13: src/main.c

```

1  #define _GNU_SOURCE
2  #include "div.h"
3  #include <errno.h>
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7
8  int EXIT_CODE = 0;
9
10 int main() {
11     char* input = NULL;
12     size_t n = 0;
13     if (getline(&input, &n, stdin) == -1) {
14         printf("%s\n", strerror(errno));
15         EXIT_CODE = 1;
16         goto free_input;
17     }
18     scan_numbers_divisible_by_three(input);
19 free_input:
20     free(input);
21     return EXIT_CODE;
22 }

```

Листинг 14: scripts/generate\_mutants.sh

```

1  #!/bin/bash
2  mutate ./src/div.c --mutantDir ./mutants --cmd "./scripts/build_with_mutant.sh MUTANT"

```

Листинг 15: scripts/run\_tests.py

```

1  import argparse
2  from typing import Iterator, Tuple
3  import subprocess
4  from sys import exit
5  from enum import Enum
6
7
8  def iter_test_samples(filename: str) -> Iterator[Tuple[str, str]]:
9      with open(filename, "r") as test_file:
10         for test in test_file.readlines():

```



```

11         test_input, test_output = map(
12             lambda x: "".join(x),
13             zip(*map(lambda x: x.split("/"), test.strip().split()))),
14         )
15         yield test_input, test_output
16
17
18 class TestingResult(Enum):
19     SUCCESS = 0
20     INCORRECT = 1
21     TIMEOUT = 2
22     ERROR = 3
23
24
25 def run_tests(*, tests_filename: str, executable: str, verbose: bool) -> TestingResult:
26     for test_input, test_output in iter_test_samples(tests_filename):
27         try:
28             output = subprocess.check_output(
29                 [executable], input=test_input.encode(), timeout=1
30             ).decode()
31         except subprocess.TimeoutExpired:
32             return TestingResult.TIMEOUT
33         except subprocess.CalledProcessError:
34             return TestingResult.ERROR
35         assert type(output) is type(test_output)
36         if output != test_output:
37             if verbose:
38                 print("Failed on input:", test_input)
39                 print("Program output:", output)
40                 print("Expected output:", test_output)
41             return TestingResult.INCORRECT
42     return TestingResult.SUCCESS
43
44
45 if __name__ == "__main__":
46     parser = argparse.ArgumentParser()
47     parser.add_argument("filename", type=str, help="Input FSM file")
48     parser.add_argument(
49         "-e", "--executable", type=str, help="Executable path", required=True
50     )
51     parser.add_argument(
52         "-v", "--verbose", action="store_true", help="Whether to print fail case"
53     )
54     args = parser.parse_args()
55     exit(
56         run_tests(
57             tests_filename=args.filename,
58             executable=args.executable,
59             verbose=args.verbose,
60         ).value
61     )

```

## Листинг 16: scripts/build\_with\_mutant.sh

```

1  #!/bin/bash
2  MUTANT=${1:-./src/div.c}
3  rm -rf build
4  cmake -S. -Bbuild -DDIV_C=${MUTANT}
5  cmake --build build

```

## Листинг 17: scripts/test\_mutants.sh

```

1  #!/bin/bash
2  SCRIPT_DIR="$( cd -- "$( dirname -- "${BASH_SOURCE[0]}" )" && /dev/null && pwd )"
3  TESTS_FILE=${1:-tests.txt}
4  PASSING_MUTANTS=${2-passing_mutants.txt}
5  > ${PASSING_MUTANTS}
6  for mutant in $(find mutants -type f -name "div.mutant.*.c"); do
7      echo "***${mutant}***"
8      sh ${SCRIPT_DIR}/build_with_mutant.sh ${mutant} &>/dev/null
9      python3 ${SCRIPT_DIR}/run_tests.py -e build/div ${TESTS_FILE}
10     exit_code=$?
11     if [[ ${exit_code} = 0 ]]; then
12         echo ${mutant} >>${PASSING_MUTANTS}
13         diff --color ./src/div.c "${mutant}"
14     elif [[ ${exit_code} = 2 ]]; then
15         echo "***Timeout***"
16     elif [[ ${exit_code} = 3 ]]; then
17         echo "***Runtime error***"
18     fi
19 done

```

## Листинг 18: scripts/list\_mutants\_with\_same\_assembler.sh

```

1  #!/bin/bash
2  MUTANTS_LIST_FILE=${1:-passing_mutants.txt}
3
4  remove_first_line() {
5      tail -n +2 "$1"
6  }
7
8  gcc -S -o div.s ./src/div.c
9
10 for mutant in $(cat "${MUTANTS_LIST_FILE}"); do
11     gcc -S -o div_mutant.s ${mutant} &>/dev/null
12     diff <(remove_first_line div.s) <(remove_first_line div_mutant.s) &>/dev/null && \
13     { echo ${mutant}; diff --color ./src/div.c ${mutant} >&2; }
14 done
15
16 rm div.s div_mutant.s

```

## Листинг 19: CMakeLists.txt

```

1  cmake_minimum_required(VERSION 3.13.4)
2  project(div LANGUAGES C)
3  set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -Wall -Wextra -pedantic -O2 -std=c99")
4
5  if (NOT DEFINED DIV_C)
6      set(DIV_C src/div.c)
7  endif()
8
9  add_executable(div ${DIV_C} src/main.c)
10 target_include_directories(div PUBLIC include)

```