

Использование конечно автоматных методов при тестировании

Выполнил Хафизов Хасан

Введение

В этой лабораторной работе для некоторой системы строится автоматное описание и её программная реализация. Далее, используя автоматное описание строятся тестовые последовательности, которые подаются на вход программной реализации, таким образом доказывается корректность реализации. Далее с помощью мутационного тестирования оценивается полнота тестирования.

Неформально описание системы

Необходимо подсчитать количество слов в некотором английском тексте, начинающихся и заканчивающихся на "s"

1. Система считывает текст посимвольно, считая пробел отдельным символом
2. Все знаки кроме "s" и пробела считаются равнозначными
3. При обнаружении подходящего слова автомат возвращает команду "++"

Описание программной реализации

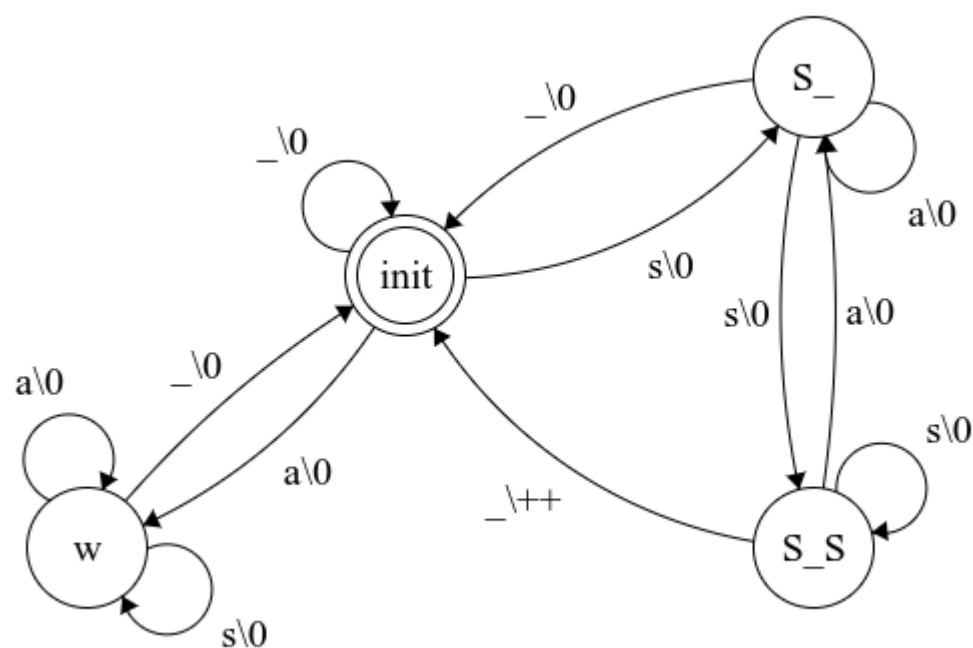
Программа реализована на языке C. По ссылке <https://github.com/xozzslip/fsm-lab> можно найти весь код, а здесь приведём листинг функциональной части программы:

```
// 4 возможных состояния
enum State {
    INIT, // начальное состояние
    S_, // в начале текущего слова была "s"
    S_S, // в текущем слове найдена вторая "s"
    W // текущее слово не начиналось с "s"

int count(char *text) {
    char current;
    int counter = 0;
    enum State state = INIT;
    while (1) {
        current = *text++;
        switch(state){
            case INIT:
                if(current == 's') {
                    state = S_;
                }
                else if(current == ' ' || current == '\0') {
                    state = INIT;
                }
            }
        }
    }
}
```

```
        else {
            state = W;
        }
        break;
    case S_:
        if(current == 's') {
            state = S_S;
        }
        else if(current == ' ' || current == '\\0') {
            state = INIT;
        }
        else {
            state = S_;
        }
        break;
    case S_S:
        if(current == 's') {
            state = S_S;
        }
        else if(current == ' ' || current == '\\0') {
            state = INIT;
            counter++;
        }
        else {
            state = S_;
        }
        break;
    case W:
        if(current == 's') {
            state = W;
        }
        else if(current == ' ' || current == '\\0') {
            state = INIT;
        }
        else {
            state = W;
        }
        break;
    }
    if (current == '\\0') {
        break;
    }
}
return counter;
}
```

Автоматное описание системы



Входные и выходные значения:

- "s" — символ "s"
- " " — пробельный символ
- "a" — любой (any) символ, кроме "s" и пробела
- "0" — выходное воздействие отсутствует
- "++" — выходное воздействие, означающее, что искомое слово обнаружено

Список состояний совпадает с состояниями в программной реализации.

Тестирование программной реализации

По автоматному описанию были построены следующие тестовые последовательности (всего 23 штуки):

ID	input	output
0	"sss "	1
1	"ss ss "	2
2	"ss s "	1
3	"ss "	1
4	"ssqs "	1
5	"ssl "	0
6	"s ss "	1
7	"s s "	0
8	"s "	0

ID	input	output
9	"sms "	1
10	"sn "	0
11	" ss "	1
12	" s "	0
13	" "	0
14	"qsss "	0
15	"lss "	0
16	"ns "	0
17	"q ss "	1
18	"q s "	0
19	"q "	0
20	"qqss "	0
21	"qqs "	0
22	"qq "	0

Программная реализация успешно проходит юнит-тесты составленные из приведённых выше тестовых последовательностей.

Процедура генерации мутантов

Мутанты генерируются с помощью библиотеки <https://github.com/mull-project/mull>.

Примеры мутантов:

- Negate mutator 1

```
if (current == '\0') // replaced == with != (32->33)
```

- Negate mutator 2

```
else if(current == ' ' || current == '\0') { // replaced == with != (32->33)
```

- Math add mutator

```
counter++; // Math Add: replaced + with -
```

Оценка полноты тестов

Всего было построено 14 мутантов, для каждого из которых было запущено по 23 теста, сгенерированных по автоматной реализации. Выпишем отношение количества сломанных тестов к их общему количеству для каждого типа.

ID	failed	score
count_0_22_2_math_add_mutator	9/23	39%
count_0_2_2_negate_mutator	14/23	61%
count_0_4_2_negate_mutator	5/23	21%
count_0_5_2_negate_mutator	3/23	13%
count_0_10_2_negate_mutator	7/23	30%
count_0_12_2_negate_mutator	2/23	9%
count_0_12_2_negate_mutator	1/23	4%
count_0_18_2_negate_mutator	9/23	39%
count_0_20_2_negate_mutator	9/23	39%
count_0_21_2_negate_mutator	1/23	4%
count_0_26_2_negate_mutator	1/23	4%
count_0_28_2_negate_mutator	2/23	9%
count_0_29_2_negate_mutator	1/23	4%
count_0_34_2_negate_mutator	9/23	39%

Каждый мутант не проходит как минимум один тест, отсюда можно сделать вывод, что полнота этих тестов достаточно высока.

Вывод

Программная реализация системы успешно проходит набор тестов, порождённый автоматным описанием системы. Набор тестов, в свою очередь, обладает свойством полноты, которое было подтверждено на этапе мутационного тестирования. Два эти свойства позволяют утверждать, что программная реализация является корректной.