Формальные методы программной инженерии

Лабораторная работа №1
“Использование конечно автоматных методов при тестировании программного обеспечения”

Токмаков Александр
группа МСП20

Введение

В этой лабораторной работе рассматривается программная реализация некоторого конечного автомата, которая проверяется с помощью набора тестов, сгенерированных методом Василевского по исходному автомату-спецификации.

Исходный код программы и скриптов для запуска тестов, а также логи запуска тестов и примеры программ, прошедших тесты при мутационном тестировании доступны в репозитории на GitHub: https://github.com/tavplubix/hse-fsm

Структура репозитория:

- do_all.sh - точка входа, запускает остальные скрипты
- results/ - сгенерированные файлы и логи тестов
- survived_mutants/ - код программ-мутантов, прошедших тесты
- остальные скрипты в корне - см. описания в основной части текста работы

Тестируемая система

Система №6 из списка. Подсчитывает слова в английском языке, оканчивающиеся на “ing”, слова разделены пробелом. При обнаружении подходящего слова выводит “1”.

В задании написано “Все знаки кроме s и пробела считаются равнозначными.”. Скорее всего, это следствие копипасты из задания №4, и имелось ввиду, что равнозначными нужно считать все знаки кроме i, n, g и пробела. Так и будем считать далее.

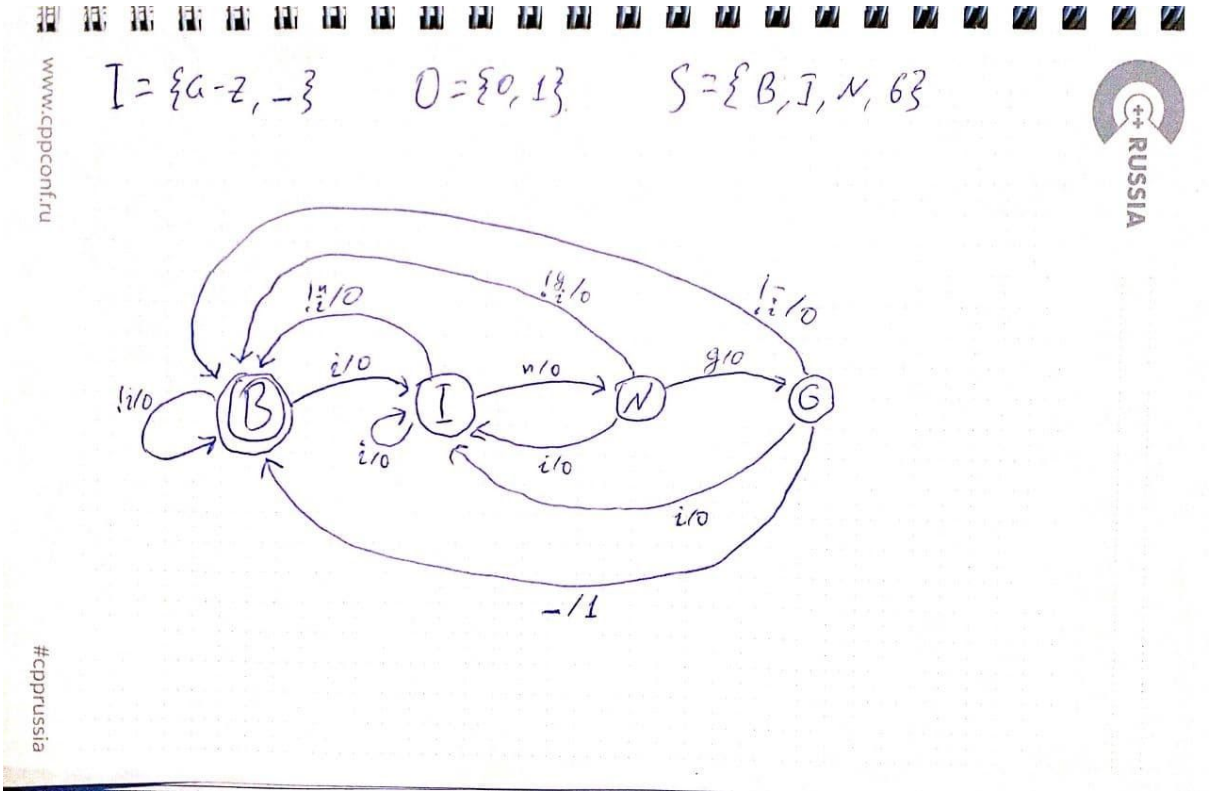
Представим её в виде конечного автомата.

В качестве множества входных символов выберем **[a-z_]**, т.е. все маленькие латинские буквы от a до z и нижнее подчеркивание, которое заменяет пробел (для удобства). Альтернативные варианты: можно добавить к алфавиту ещё заглавные буквы и остальные пробельные символы (или даже весь ASCII или Unicode), но смысла в этом мало, а одинаковых переходов между состояниями слишком много; также можно использовать специальный символ, означающий “что угодно, кроме i, n, g и пробела”, но это недостаточно интересно (получается слишком маленький автомат).

Множество выходных символов будет состоять из двух символов: **0** и **1**. 1 должен выводиться, когда нашлось нужное слово во входных данных, а 0 - во всех остальных случаях (т.к. автомат должен что-то вывести на каждый входной символ; можно считать 0 “пустым” символом).

Множество состояний: **{BEGIN, I, N, G}**. BEGIN - начальное состояние, остальные соответствуют ситуации, когда мы уже нашли соответствующий префикс строки “ing” во входных данных и ожидаем следующего символа.

Получается вот такой автомат:



Описание автомата в формате FSMtest генерируется с помощью **gen_fsm.py**, сгенерированное описание находится в **results/fsm.txt**.

Генерация тестов

Тесты генерируются скриптом **curls.sh**, который загружает описание автомата на http://fsmtestonline.ru/ и, используя API сайта, валидирует автомат, проверяет его свойства и скачивает набор тестов, сгенерированных методом **black_box/w** для **m=4**. Как можно видеть из вывода этого скрипта (**results/do_all.out.log**), автомат детерминированный, полностью определенный, связный и приведённый (впрочем, это сразу было видно по автомату). Полученный набор тестов записан в **results/fsm.txt.tests**.

Лирическое отступление (или что бы ни делать, лишь бы не писать текст отчёта). Понятно, что этот сайт не предполагает наличия публичного API, но, тем не менее, API сайта очень странное. Первое, что бросается в глаза - на сервер загружаются произвольные файлы с произвольными именами (по крайней мере, выглядит так), вместо уникального ключа в последующих запросах к API используется используется имя файла, которое указал пользователь. При таком API у некоторых пользователей может возникнуть желание использовать имена вроде ../../../../etc/passwd или index.php (хотя на подозрительные расширения файлов сервер ругается, выглядит это очень ненадёжно). Или загрузить кучу тяжёлых файлов (они же просто остаются на сервере?). Или сделать ещё что-то интересное.

Ещё несколько небольших странностей:

- На запрос вроде http://fsmtestonline.ru/validate_fsm?input_file=fsm.txt сервер отвечает джсоном с опечаткой: {"input_file": "fsm.txt_conv", "fsm_des**c**": "fsm.txt_des**k**"}. Далее во всех запросах значение параметра fsm_desc тоже получается с опечаткой.
- Некоторые текстовые поля внутри json-ответов сервера содержат html-тег

P.S. Прошу не воспринимать это как придирки, всё же свою основную работу сайт прекрасно выполняет :).

Программная реализация

В виде функции в файле **fsm.py**. Можно запускать программу через **fsm_main.py**, передав входную последовательность символов на stdin, выходная последовательность печатается на stdout.

Код программной реализации:

```
def get_fsm_output(fsm_input):
    look_for = 'ing '
    s = 0

    res = ''
    for c in fsm_input:
          o = '0'
          if c == look_for[s]:
          if s == 3:
                o = '1'
                s = 0
          else:
                s = s + 1
          elif c == look_for[0]:
          s = 1
          else:
          s = 0
          res += o

    return res
```

Тестирование и мутанты

Тесты для программы **fsm_main.py** запускаются скриптом **run_tests.py**, который принимает два аргумента: имя файла со сгенерированными тестами и имя исполняемого файла. Все тесты проходят, ошибок не найдено (логи в **results/tests.out.log**). Также можно запустить тесты для функции с помощью модуля **unittest** (логи в **results/unit_tests.out.log**).

Мутанты генерируются с помощью модуля MutPy, этот же модуль запускает юнит-тесты для каждого сгенерированного мутанта. Изменения в исходном коде и результаты мутационного тестирования записаны в **results/mut_tests.out.log**.

```
[*] Mutation score [5.52060 s]: 85.7%
   - all: 35
   - killed: 29 (82.9%)
   - survived: 5 (14.3%)
   - incompetent: 0 (0.0%)
   - timeout: 1 (2.9%)
```

Из 35 сгенерированных мутантов 5 успешно прошли тесты, полнота 85.7%. Исходный код прошедших тесты мутантов находится в **servived_mutants/fsm{1-5}.py**.

Все прошедшие тесты мутанты довольно похожи и говорят о том, что тесты не покрывают случаи достаточно длинных входных последовательностей, когда строка ‘ing’ и её префиксы могут встречаться во входных данных много раз.