Поддержка фаззинга в языках программирования

Сергей Бронников, Tarantool

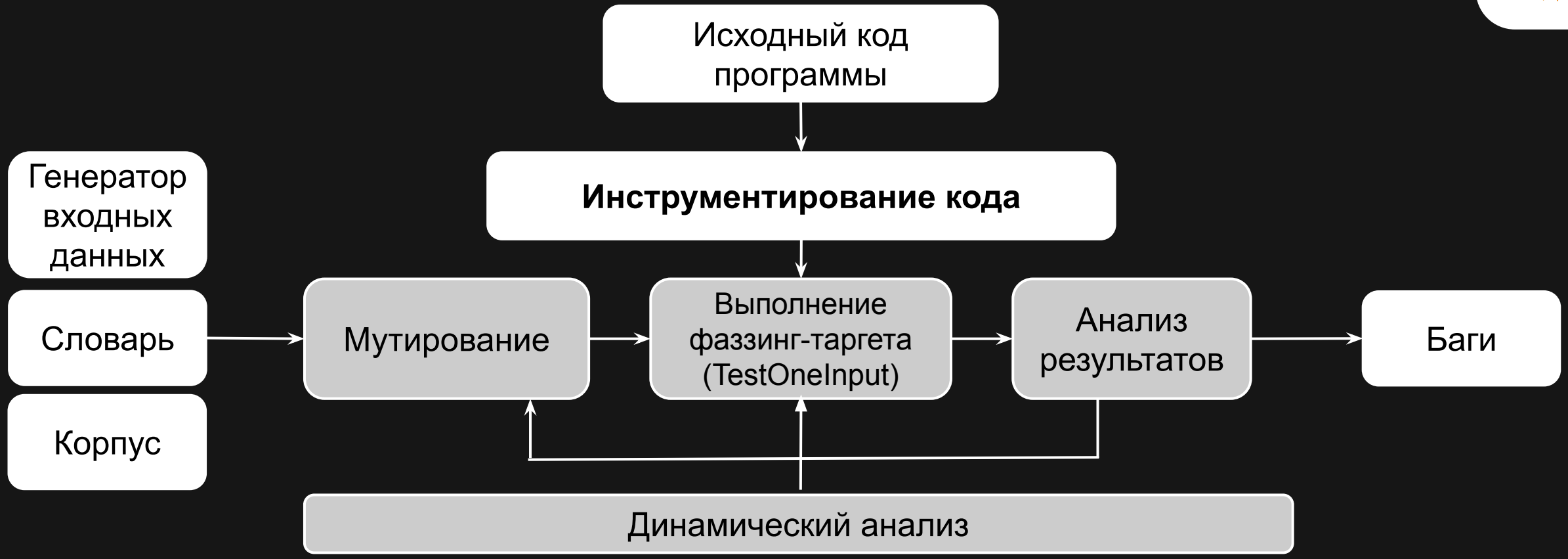
План

- Рандомизированное тестирование
- Инструментация исходного кода
- Реализация инструментации в ЯП
- Выводы
- Наблюдения
- Вопросы

Рандомизированное тестирование

- Входные данные генерируются (псевдо)случайным методом.
- Основные подходы:
  - Тестирование с помощью свойств (property-based testing).
  - Фаззинг (обычно с инструментацией).
- Обратная связь может присутствовать или нет (“чёрный”, “белый” или “серый” ящик).

Фаззинг: принципиальная схема



Протестируем функцию сложения

двух целых чисел с использованием PBT и фаззинга

Это функция сложения целых чисел (с сюрпризом)

```
def add(x, y):
    if x == 100 and y == 200:
        return y - x    # сюрприз!
    return x + y + y
```

Тестирование с помощью свойств: тесты Hypothesis

```
# Два случайных целых числа.
@given(arg1=st.integers(), arg2=st.integers()
def test_add_hypothesis(self, arg1, arg2):
    self.assertEqual(add(arg1, arg2), add(arg2, arg1))
```

Тестирование с помощью свойств: результат

```
$ python3 -m unittest add_tests.TestSuiteAdd.test_add_hypothesis
...
Ran 1 test in 0.113s
OK
```

Тестирование с помощью фаззинга: тесты Atheris (1/2)

```
def TestOneInput(input_bytes):
    fdp = atheris.FuzzedDataProvider(input_bytes)
    arg1 = fdp.ConsumeInt()    # Генерируем знаковое целое.
    arg2 = fdp.ConsumeInt()    # Генерируем знаковое целое.
    self.assertEqual(add(arg1, arg2), add(arg2, arg1))
```

Тестирование с помощью фаззинга: тесты Atheris (2/2)

```
def test_add_atheris(self):
        atheris.Setup(sys.argv, TestOneInput)
        atheris.Fuzz()              # Запускаем фаззинг.
```

Тестирование с помощью фаззинга: результат

```
$ python3 -m unittest add_tests.TestSuiteAdd.test_add_atheris
AssertionError: 100 != 300
...
Ran 1 test in 0.137s
```

Рандомизированное тестирование **эффективнее** с обратной связью

* Making Software Dumberer, Tavis Ormandy, Google, Inc.

Цели и виды инструментации исходного кода

Информация о потоках исполнения (Control-Flow), чтобы понять какие тест-кейсы полезны:

- Покрытие базовых блоков (Basic Block Coverage).
- Покрытие ветвей (Edge Coverage).
- Покрытие N-грам (N-gram coverage).
- Контекстное покрытие.

Информация о потоках данных исполнения (DataFlow), полезная для мутаций.

Извлечение аргументов из операторов сравнения, деления и т.д.

Подробнее: Clang SanitizerCoverage C API

Реализация инструментации специфична для каждого языка программирования

Рассмотрим разные реализации инструментации кода в ЯП

libFuzzer (C/C++)

- Clang позволяет инструментировать как потоки данных так и потоки управления.
- Опция -fsanitize=fuzzer разворачивается в набор опций для инструментации компилятором.
- State of the art среди языковых инструментаций.
- libFuzzer и механизмы инструментации доступны “из коробки” с Clang 6+.
- В GCC возможности инструментации скромнее.

Подробнее: Clang SanitizerCoverage C API

Atheris (Python)

- Python использует стековую виртуальную машину для выполнения байткода.
- Atheris - расширение для CPython, реализует обратную связь для libFuzzer и Python API к libFuzzer.
- Atheris использует комбинацию из двух видов инструментации:
    - инструментация на основе перехватчика (хука)
    - инструментация байткода (Python 3.8+)

Подробнее: How the Atheris Python Fuzzer Works

Atheris (Python): инструментация с помощью хука

- Atheris реализует функцию-хук, которая фиксирует номера исполняемых строк кода. Функция использует Clang API `__*sanitizer_cov_pcs_init()*` и `__*sanitizer_cov_8bit_counters_init()*`.
- В начале исполнения скрипта Atheris регистрирует функцию в *sys.settrace(tracefunc)*.
- При исполнении функция вызывается для **каждой** выполняемой строки кода. Негативно сказывается на скорости выполнения (фаззинга).

Atheris (Python): инструментация байткода 1/2

- В Python 3.8 появилась возможность трассировки байткода, что позволило дополнительно инструментировать.
- Инструментация байткода *COMPARE_OP* позволяет извлекать аргументы из операций сравнения и передавать их в libFuzzer с помощью Clang API `__sanitizer_weak_hook_memcmp()` и `__sanitizer_cov_trace_cmp8()`.

Atheris (Python): инструментация байткода 2/2

```
In [5]: def less(a, b):
   ...:   return a < b
   ...:
```

```
In [6]: dis.dis(less)
  1           0 RESUME                   0

  2           2 LOAD_FAST                0 (a)
              4 LOAD_FAST                1 (b)
              6 COMPARE_OP               2 (<)
             10 RETURN_VALUE

In [7]:
```

Jazzer (Java) 1/2

- Использует библиотеку libFuzzer.
- Инструментирует байткод виртуальной машины (JVM).
- Инструментация происходит во время выполнения, а не во время сборки.
- Инструментация заключается в добавлении байткода с уникальным ID в начало каждого базового блока.

Jazzer (Java) 2/2

- Фаззер состоит из двух основных компонентов:
  - Jazzer-драйвер, бинарный файл, скомпонованный с libFuzzer, который непосредственно запускает цель с помощью Java Native Interface (JNI).
  - Jazzer-агент, запускается в той же ВМ и выполняет инструментацию во время выполнения. Отправляет собранные данные в libFuzzer.

Ruzzy (Ruby)

- Ruby интерпретируемый язык, выполняется в стековой виртуальной машине.
- Ruzzy - это стороннее C-расширение для Ruby, реализует обратную связь и Ruby API, использует libFuzzer.
- Для инструментации используется механизм перехвата (хуки) модуля TracePoint, потому что он в отличие от модуля Coverage имеет C API.
- Трудности с использованием Ruby API для обратной связи.

Подробнее: **Design and Implementation of a Coverage-Guided Ruby Fuzzer**, Matt Schwager, Dominik Klemba, Josiah Dykstra

Go

- Сторонняя библиотека dvyukov/go-fuzz, на основе libFuzzer.
- В Go 1.14, 1,18, 1.19 последовательно появилась нативная поддержка фаззинга на основе libFuzzer.
- Инструментация потока исполнения.
- Инструментация потока данных:
  - Перехват аргументов операций сравнения строк - *strings.EqualFold()* и передача в libFuzzer с помощью `__*sanitizer_cov_trace_{,const_}cmp{1,2,4,8}*`

Подробнее: Golang Fuzzing: Key Improvements in Go Fuzzing (Golang 1.19)

Заключение

- Инструментация повышает эффективность фаззинга.
- Рассмотрели виды инструментации кода.
- Рассмотрели виды реализаций для разных ЯП.

Наблюдения

- Библиотеки не являются частью поставки языка программирования, хотя очень тесно с ним интегрированы и повышают безопасность и качество проектов, написанных на этом языке. Исключение - Go и C/C++.
- Фаззинг-библиотеки широко используются в РБПО. Все были реализованы волонтёрами или компаниями из индустрии ИБ. Вы им доверяете?
- Эффективность языковых интеграций ничем не подтверждается, нет результатов в FuzzBench.

Вопросы?

Сергей Бронников

t.me/sqaunderhood