Оптимизация читаемости тестов порождаемых при символьных вычислениях

И.А. Якимов, Кузнецов А.С.

Институт Космических и Информационных Технологий, Сибирский Федеральный Университет

Проблема

- Тестирование занимает **~50%** времени разработки*
- Автоматически сгенерированные тесты **трудно проверять вручную**

*Hambling B. Realistic and Cost-effective Software Testing.
Kelly, Management and Measurement of Software Quality, UNICOM SEMINARS, Middlesex, UK, 1993, pp. 95-112.

Идея

- Подстроить входные данные под **модель естественного языка**
- **Биграммная модель** дает оценку вероятности следования B после A для пары символов AB
- **Пример**: появление в тексте пары «is» более вероятно чем «#@»

Идея

- Данная идея впервые предложена реализована в работе Afshan и др.* в контексте генерации тестов на основе **мета-эвристического поиска** (SBST) — оценка читаемости встроена в функцию приспособленности.

Afshan S., McMinn P., Stevenson M.
Evolving readable string test inputs using a natural language model to reduce human oracle cost.
International Conference on Software Testing, Verification and Validation, 2013.

Идея

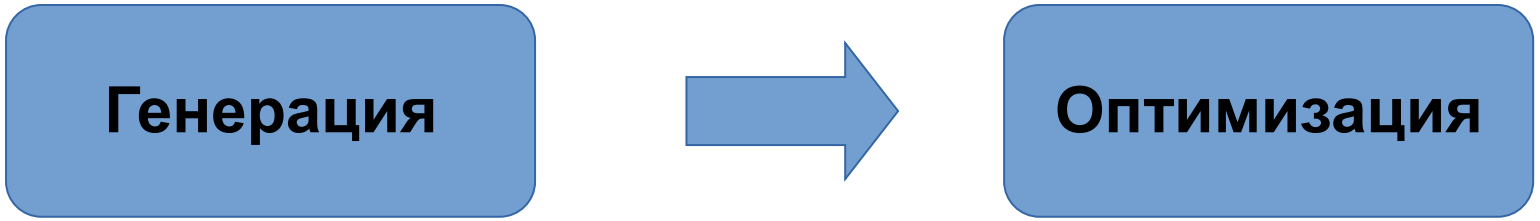
- В нашей работе биграммная модель используется для оптимизации читаемости в контексте **динамических символьных вычислений** (DSE) — оптимизация проводится по ограничению пути

Оценка читаемости

$$N = \hat{P}\left(c_1^n\right)^{1/n} \qquad (2)$$

Задача

Максимизация степени читаемости строки, с сохранением случайной природы порождаемых при этом слов


Генерация
Оптимизация

Фаза генерации

- По мере выполнения символьных вычислений формируется ограничение пути (PC), характеризующее **класс эквивалентности** входных данных, проводящих программу по данному пути

Фаза оптимизации

- **Оптимизация читаемости** происходит **после** завершения символьных вычислений, когда PC уже сформировано

Мотивация

Для каждого отдельного пути выполнения множество решений PC может включать входные векторы со строками, содержащими **печатные символы** и **целые слова**

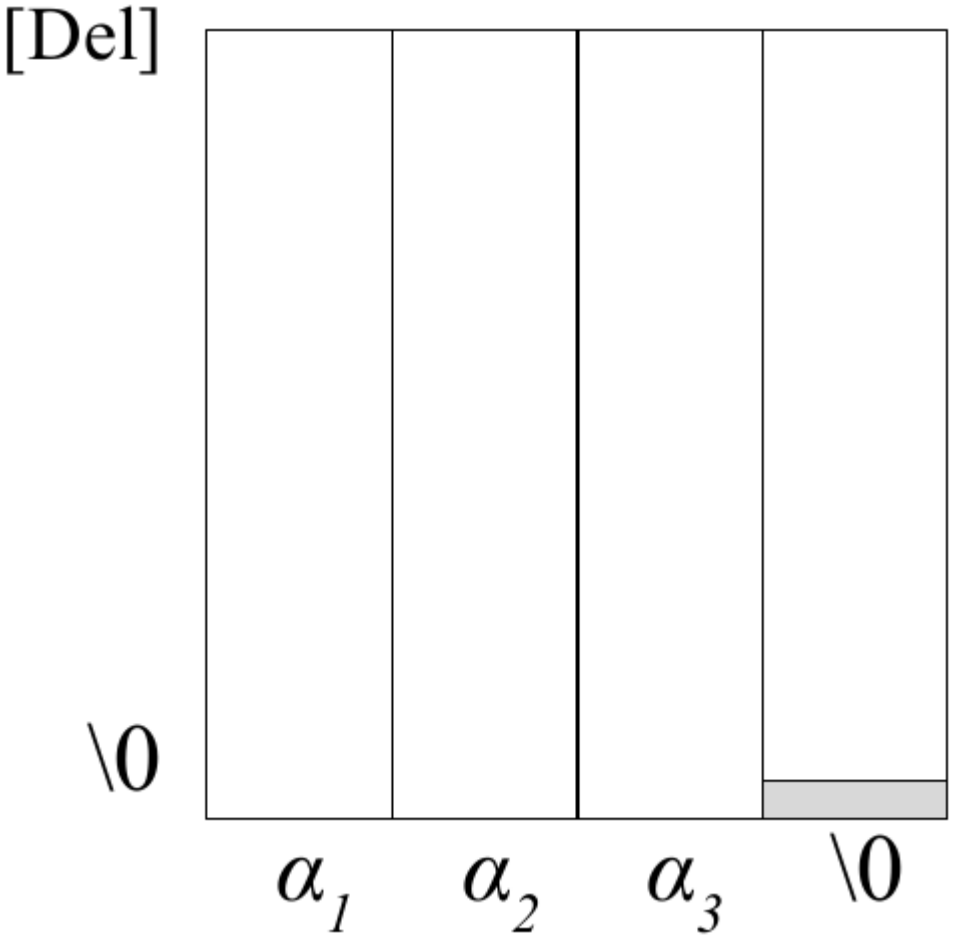
Алгоритм

Метод **последовательно конкретизирует** значения входящих в строки символов, по мере возможности выстраивая их согласно **биграммной модели**

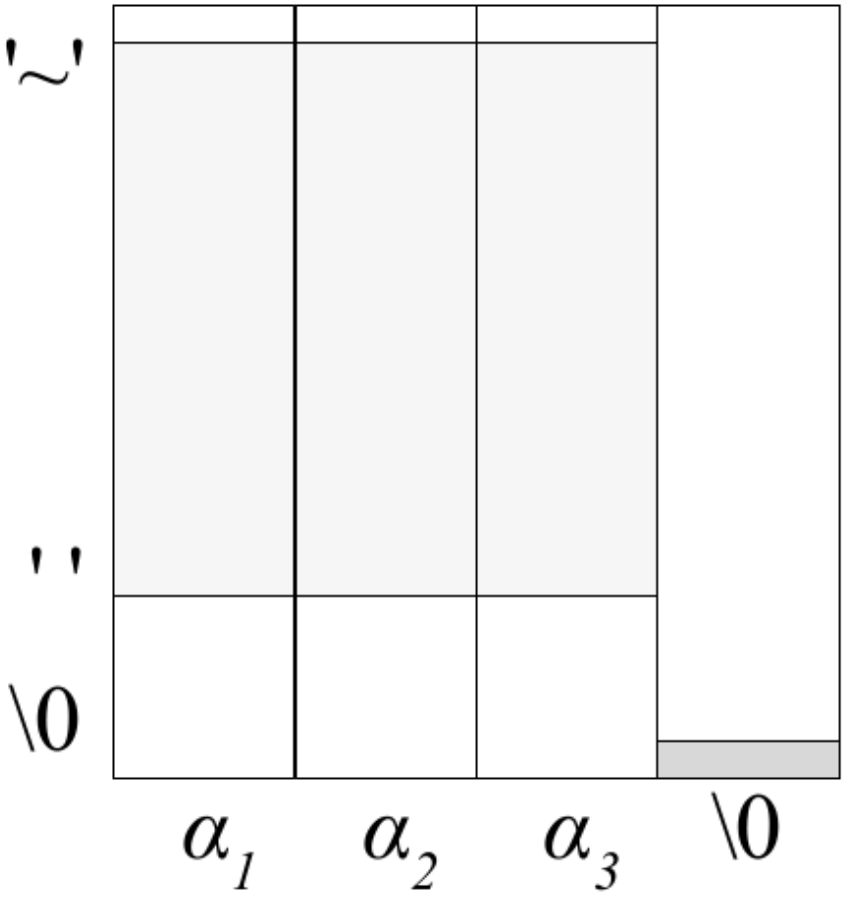
Пример - strlen

Разберем алгоритм на примере функции **strlen**

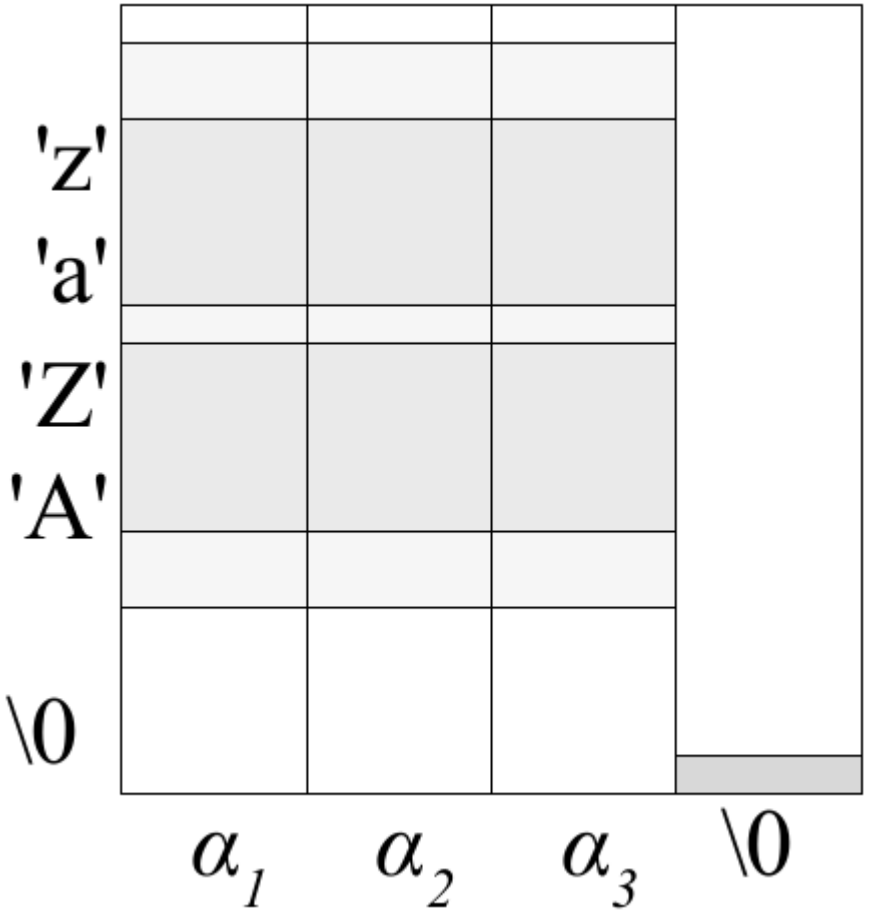
Пример - strlen



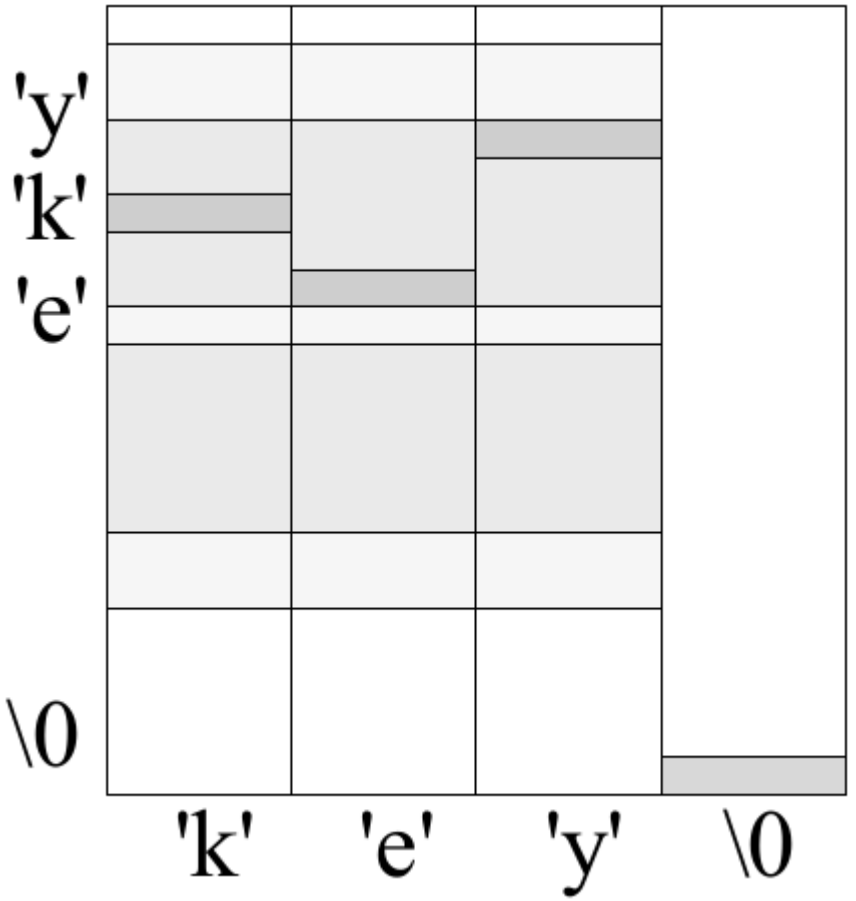
Пример - strlen



Пример - strlen



Пример - strlen



Консервативность алгоритма

- **Оптимизированные тесты работают «также» как и неоптимизированные:** перед тем как наложить очередное ограничение алгоритм проверяет, не нарушит ли оно целостности PC.

Эксперимент: 12 функций из репозитория Linux

strlen

strnlen

strcmp

strncmp

sysfs_streq

strcpy

strncpy

strcat

strncat

strstr

strnstr

strpbrk

Результаты

- **Покрытие** в среднем 95% инструкций, для 9 из 12 100%
- **Степень читаемости*** сопоставима со списком Top-100 английских слов: 0,08 (σ = 0,03) против 0,10

*нормализованная оценка вероятности принадлежности строки корпусу языка полученная при помощи биграммной модели

strpy

- Посмотрим как работает генерация и оптимизация на примере функции **strcpy**

Неоптимизированный вывод

```
&"\x01\x01\x01\x01\x01"{\0}
&"\x01\x01\x01"{\0,\0,\0} :=>
&"\x01\x01\x01"{\0,\x01,\0}
```

Переводим в печатный диапазон

&"aaaaa"{\0}

&"aaa"{\0,p,\0} :=>

&"aaa"{\0,a,\0}

Добавляем биграммную модель

&"athes"{\0}

&"ath"{\0,p,\0} :=>

&"ath"{\0,s,\0}

Случайно выбираем первый символ

&"kesth"{\0}

&"pre"{\0,p,\0} :=>

&"pre"{\0,h,\0}

*возможны зацикливания: «thesthes...»

Используем метод рулетки

&"fmf"{\0,\x01,\x01}

&"emslo"{\0} 5 :=>

&"emslo"{\0}

Выбираем буквы случайно

&"jbg"{\0,\x01,\x01}

&"ttabn"{\0} :=>

&"ttabn"{\0}

Генератор

- Рабочее название - Huntl
- Компиляторная инфраструктура **LLVM**
- SMT-решатель **CVC4**
- Биграммная модель по корпусу из ~183 млн. слов*
- Репозиторий проекта: https://github.com/IvanYakimov/huntl

* Jones M. N., Mewhort D.J.K. Case-sensitive letter and bigram frequency

Итоги

- Новый метод оптимизации читаемости тестов в контексте DSE
- Успешная апробация генератора на 12-ти строковых функциях библиотек Linux
- Может быть встроен в другие генераторы тестов

Спасибо за внимание

Иван Якимов, ivan.yakimov.research@yandex.ru

Дополнительные слайды

SBST

При использовании SBST цель тестирования (например, покрытие кода) формулируется в виде **функции приспособленности**, отображающей входные данные на некоторый количественный показатель. Генерация тестов сводится к **многократному запуску** программы с **подстройкой входных** данных для оптимизации функции приспособленности до тех пор, пока не будет достигнута поставленная цель тестирования. Для **улучшения читаемости** значение N включается в функцию приспособленности.

Экспериментальные данные

Функ	#	*Арг*	*Покр*	*Нет*	*Баз*	*Прос*	*СП*	*Рул*	*Случ.*
M			**95%**	**-**	**0,07**	**0,08**	**0,08**	**0,07**	**0,06**
σ			**8%**	**-**	**0,03**	**0,03**	**0,03**	**0,03**	**0,02**
strlen	5	[6][6]	100%	-	0,08	0,10	0,10	0,09	0,08
strnlen	6	[6][6]5	100%	-	0,08	0,10	0,10	0,09	0,08
strcmp	15	[6][6]	100%	-	0,04	0,05	0,05	0,04	0,04
strncmp	16	[6][6] 5	100%	-	0,04	0,05	0,05	0,04	0,04
sysfs_ streq	39	[6][6]	100%	-	0,05	0,07	0,06	0,06	0,05
strcpy	35	[6][6]	100%	-	0,08	0,10	0,10	0,09	0,07
strncpy	36	[6][6] 5	100%	-	0,08	0,10	0,10	0,09	0,07
strcat	40	[10][5]	100%	-	0,07	0,08	0,08	0,07	0,06
strncat	41	[10] [5]4	100%	-	0,07	0,08	0,08	0,07	0,06
strstr	19	[6][3]	90%	-	0,12	0,14	0,14	0,13	0,11
strnstr	4	[6][3]2	80%	-	0,09	0,10	0,10	0,09	0,07
strpbrk	10	[6][6]	80%	-	0,03	0,03	0,03	0,03	0,03

Оценка вероятности принадлежности строки корпусу языка

$$\hat{P}\left(c_1^n\right) \approx \prod_{i=1}^{n} P\left(c_i | c_{i-1}\right) \quad (1)$$

Метод рулетки

$$S_b = P(b)/T, \text{где } T = \sum_{1}^{n} P(c_i|b) \qquad (3)$$

Сужение

Процедура Сужение(граф памяти M)

Для каждого узла A графа памяти M

Если A — char[n], то для каждого i = 1...n:

Пусть a_i = i-й элемент массива A, пусть printable = Проба(' ' ≤ a_i ≤ '~')

Если printable, то Проба(('A' ≤ a_i ≤ 'Z') \/ ('a' ≤ a_i ≤ 'z'))

Конкретизация

Процедура Конкретизация(граф памяти M)

Для каждого узла A графа памяти M

Если A — char [n], то

Если a_1 — символьный, то

Проба(a_1 = alpha), где alpha — произвольная буква

Для каждого i = 1...n-1:

Пусть a_i a_{i+1} — пара соседних элементов массива, пусть fst = Знач(ai)