Полнота тестирования ПО

Software Testing 102

Марат Ахин

Санкт-Петербургский государственный политехнический университет

2013

Quiz

Recap

Свойства теста

What's up, Doc? ©

- Проблема тестовых входных данных
- Проблема наблюдаемости
- Проблема «останова»
- Проблема тестового оракула

Содержание

Проблема «останова» в тестировании

Проблема останова

По заданному алгоритму и входным данным определить, завершится ли за конечное время его выполнение

При чем здесь тестирование?!

Проблема «останова» в тестировании

- Алгоритм $\Leftrightarrow$ процесс тестирования
- Входные данные $\Leftrightarrow$ тестируемая программа

В подавляющем большинстве случаев процесс тестирования является бесконечным

Проблема «останова» в тестировании

- Мы не можем позволить себе тестировать бесконечное время
  - Слишком долго
  - Слишком дорого
  - Слишком странно

Что же делать?

Проблема «останова» в тестировании

Останавливать процесс тестирования «вручную»

Когда?

- У нас кончилось время и/или деньги на тестирование
- Мы протестировали ПО достаточно хорошо

Проблема «останова» в тестировании

Проблема «останова» в тестировании

По заданному набору тестов и программе определить, протестировали ли мы ее достаточно хорошо

Что такое – достаточно хорошо?

Тестовое покрытие

- Мы протестировали программу достаточно хорошо, когда мы нашли большую часть ошибок в программе
- Чтобы найти ошибку, необходимо обеспечить выполнение трех основных свойств
- Обеспечение достижимости (reachability) и порчи (corruption) требует, чтобы мы выполнили определенный участок кода с определенными входными данными

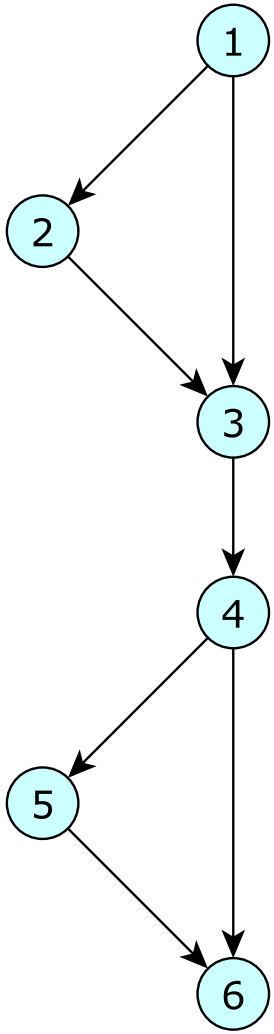
Качество тестирования можно оценить через тестовое покрытие

Виды тестового покрытия

- Выделяют два основных вида покрытия
  - Покрытие потока управления
  - Покрытие потока данных
- Они работают с такими понятиями, как граф потока управления (CFG) и граф потока данных (DFG)

Надеюсь, что все знают, что такое CFG и DFG?

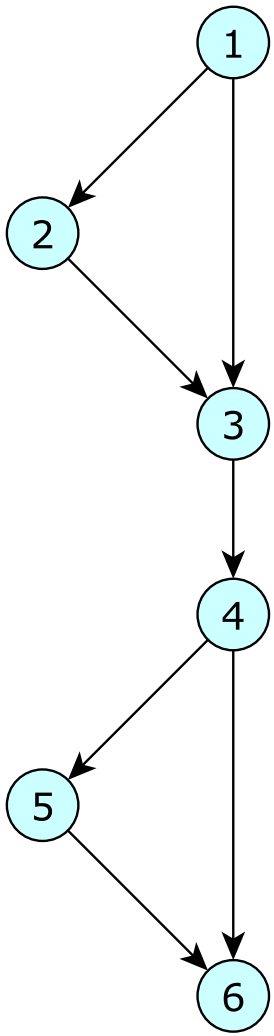
Покрытие потока управления



Как мы можем покрыть данный CFG?

- По узлам
- По дугам
- По условиям
- По путям
- ...

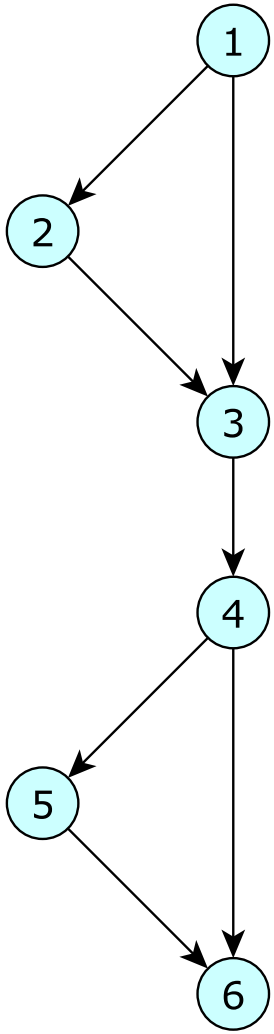
Покрытие потока управления



Как мы можем покрыть данный CFG?

- По узлам
- По дугам
- По условиям
- По путям
- ...

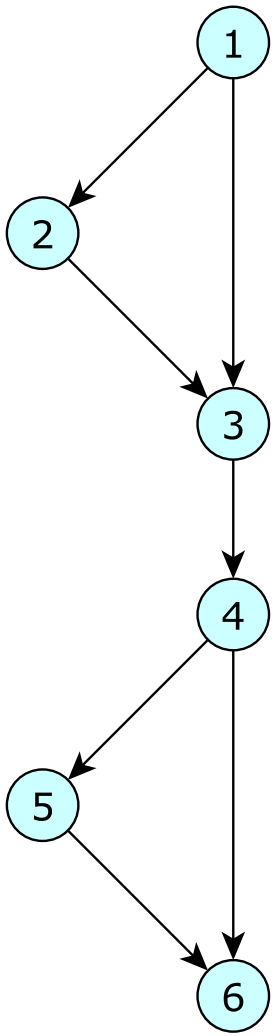
Покрытие потока управления



Как мы можем покрыть данный CFG?

- По узлам
- По дугам
- По условиям
- По путям
- ...

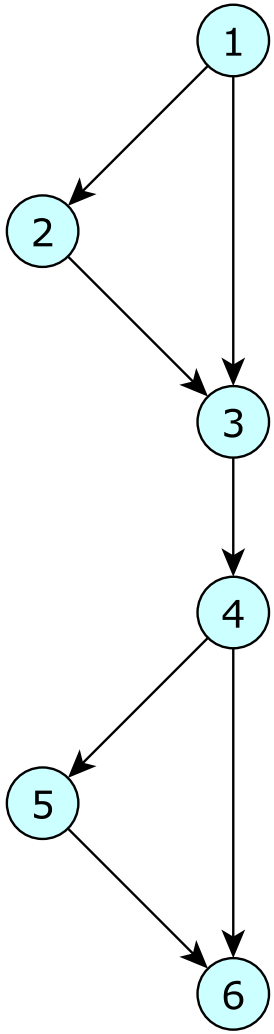
Покрытие потока управления



Как мы можем покрыть данный CFG?

- По узлам
- По дугам
- По условиям
- По путям
- ...

Покрытие потока управления

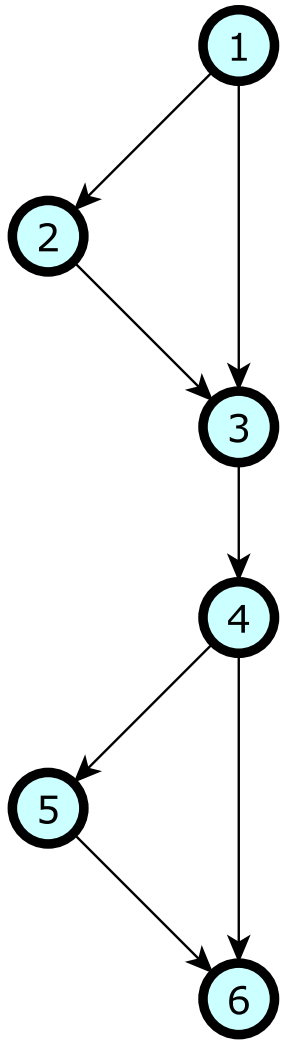


Как мы можем покрыть данный CFG?

- По узлам
- По дугам
- По условиям
- По путям
- ...

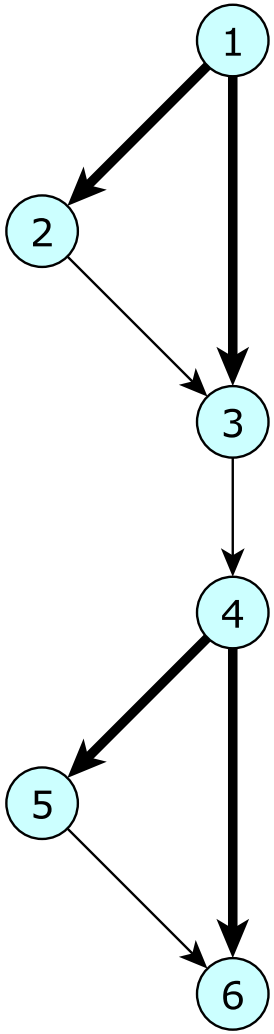
Покрытие операторов программы

- Каждый узел CFG был пройден в процессе тестирования хотя бы один раз
- Самый слабый способ оценки тестового покрытия
- Сколько тестов требуется для того, чтобы обеспечить полное покрытие операторов программы для следующего примера?



Покрытие ветвлений программы

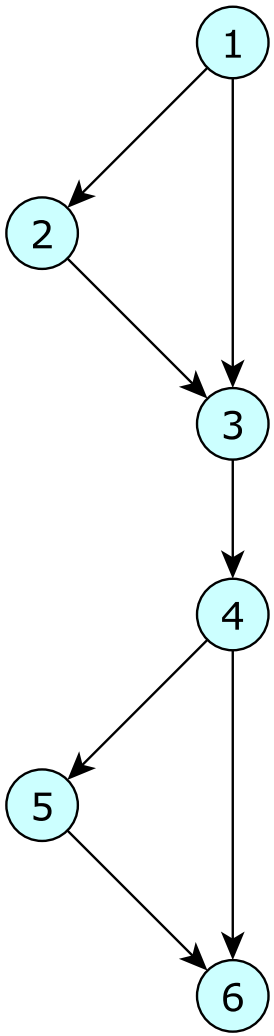
- Каждая ветка программы была пройдена хотя бы один раз
- Несколько более сильный способ оценки покрытия
- Сколько тестов требуется для того, чтобы обеспечить полное покрытие ветвлений программы для следующего примера?



Покрытие ветвлений программы

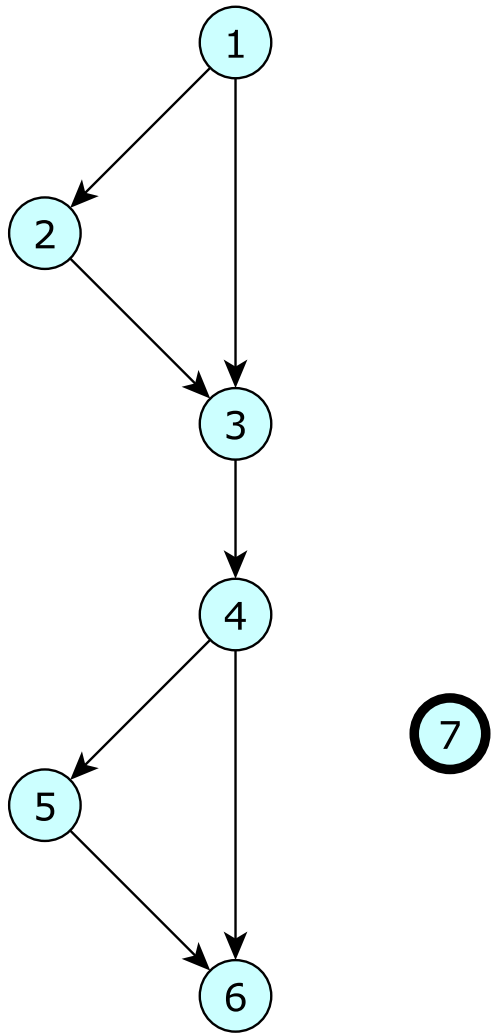
Как соотносятся между собой покрытия операторов и ветвлений?

1. Никак
2. Покрытие операторов включает покрытие ветвлений
3. Покрытие ветвлений включает покрытие операторов



Покрытие ветвлений программы

- Никак
- Почему покрытие ветвлений не включает в себя покрытие операторов?
  - Потому что в программе может присутствовать «мертвый код»



Покрытие условий программы

- Каждое ветвление может выполняться по различным причинам
- При покрытии условий программы мы требуем, чтобы все условия ветвлений хотя бы один раз приняли значение `true` и `false`

```
1 if (p != q && (p == null || !p.equals(q))) {
2     ...
3 }
```

Как соотносятся между собой покрытия ветвлений и условий?

Покрытие ветвлений и условий программы

- Комбинация соответствующих покрытий
  - Полностью покрывает их
- Можно ли предложить что-то более сильное?

```
1 if (p != q && (p == null || !p.equals(q))) {
2     ...
3 }
```

Модифицированное покрытие ветвлений и условий

- Также известное как MC/DC
- Является одним из обязательных условий при тестировании ПО на уровень A в рамках DO-178B

- Чем оно отличается от обычного покрытия ветвлений и условий программы?

```
1 if (p != q && (p == null || !p.equals(q))) {
2     ...
3 }
```

Модифицированное покрытие ветвлений и условий

Каждое условие независимо повлияло на выполнение программы

Как это проверить?

Изменить **одно** условие и проверить, изменилось ли ветвление

```
1 if (p != q && (p == null || !p.equals(q))) {
2     ...
3 }
```

Полное покрытие условий программы

- Полный перебор всех возможных **комбинаций** условий всех возможных ветвлений
- Сколько вариантов необходимо перебрать, чтобы получить полное комбинационное покрытие для следующего примера?

```
1 if (p != q && (p == null || !p.equals(q))) {
2     ...
3 }
```

Покрытие путей программы

- Мы требуем, чтобы все возможные пути программы были выполнены хотя бы один раз

Как данное покрытие соотносится с полным покрытием условий?

- Обычно считается самым сильным типом покрытия потока управления
- Его можно было бы использовать, если бы не...

Циклы и рекурсия

Покрытие путей программы

- Мы требуем, чтобы все возможные пути программы были выполнены хотя бы один раз

Как данное покрытие соотносится с полным покрытием условий?

- Обычно считается самым сильным типом покрытия потока управления
- Его можно было бы использовать, если бы не...

Циклы и рекурсия

Покрытие путей программы

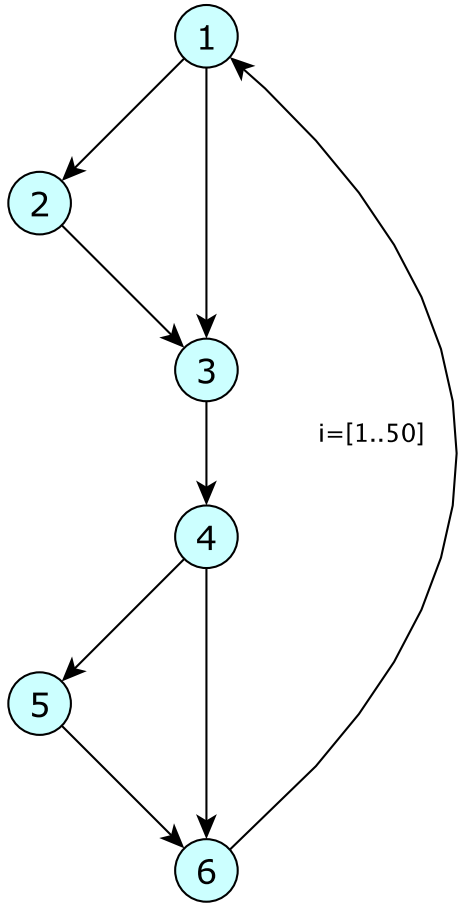
- Мы требуем, чтобы все возможные пути программы были выполнены хотя бы один раз

Как данное покрытие соотносится с полным покрытием условий?

- Обычно считается самым сильным типом покрытия потока управления
- Его можно было бы использовать, если бы не...

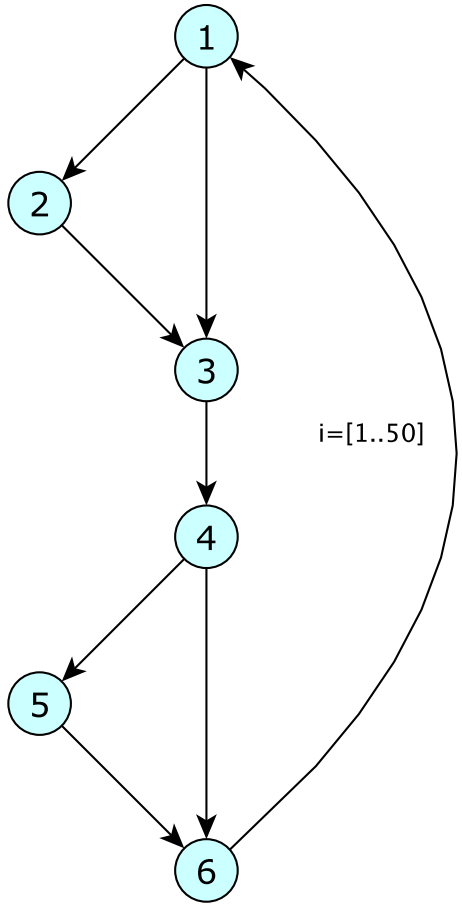
Циклы и рекурсия

Покрытие путей программы



4^{50} возможных путей

Покрытие путей программы



4^{50} возможных путей

Покрытие путей программы

- Для борьбы с этим используют несколько подходов
- Один из вариантов
  - Требуем, чтобы тело цикла было выполнено
    - 0
    - 1
    - 2
    - k
    - max

Объясните, почему выбраны именно эти варианты

Покрытие потока данных

Что еще можно сделать?

- Можно вспомнить о том, что программы работают не просто так
- С этой точки зрения для тестирования представляет интерес анализ таких путей выполнения программы, на которых активно работают с данными

Сперва вспомним несколько определений

Покрытие потока данных

Что еще можно сделать?

- Можно вспомнить о том, что программы работают не просто так
  - Основная цель работы любой программы – работа с данными
- С этой точки зрения для тестирования представляет интерес анализ таких путей выполнения программы, на которых активно работают с данными

Сперва вспомним несколько определений

Покрытие потока данных

Что еще можно сделать?

- Можно вспомнить о том, что программы работают не просто так
  - Основная цель работы любой программы – работа с данными
- С этой точки зрения для тестирования представляет интерес анализ таких путей выполнения программы, на которых активно работают с данными

Сперва вспомним несколько определений

Определения и использования переменных

Определение переменной (Def)

Оператор программы, в котором значение переменной `v` может быть изменено

- `v = 42`
- `f(&v, ...)`
- `scanf(fmt, &v)`

Использование переменной (Use)

Оператор программы, в котором значение переменной `v` влияет на выполнение программы тем или иным способом

- `q = v + 2`
- `g(v, ...)`
- `if (v != NULL) ...`

Определения и использования переменных

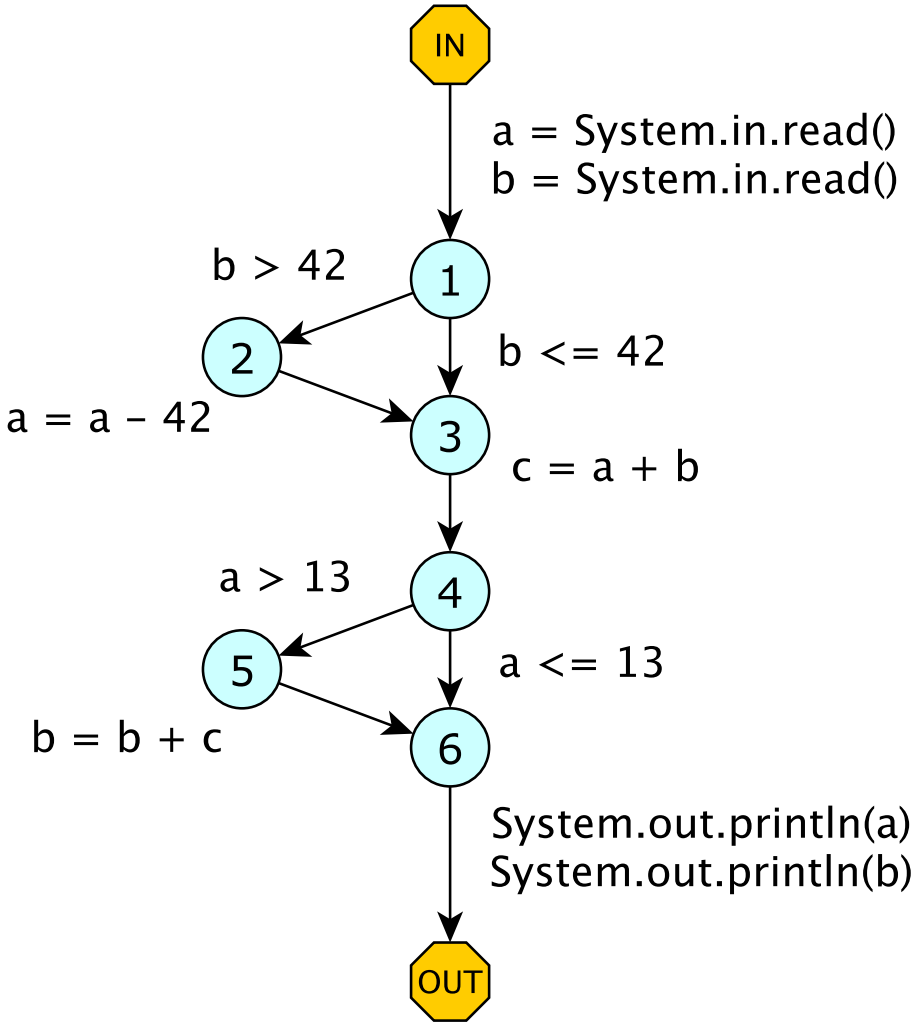
Def-Use Chain

Пара (d, u) операторов программы, для которой выполняются следующие условия

- d – определение переменной v
- u – использование переменной v
- между d и u существует хотя бы один путь, на котором переменная v не переопределяется

Рассмотрим данные определения на примере

Пример

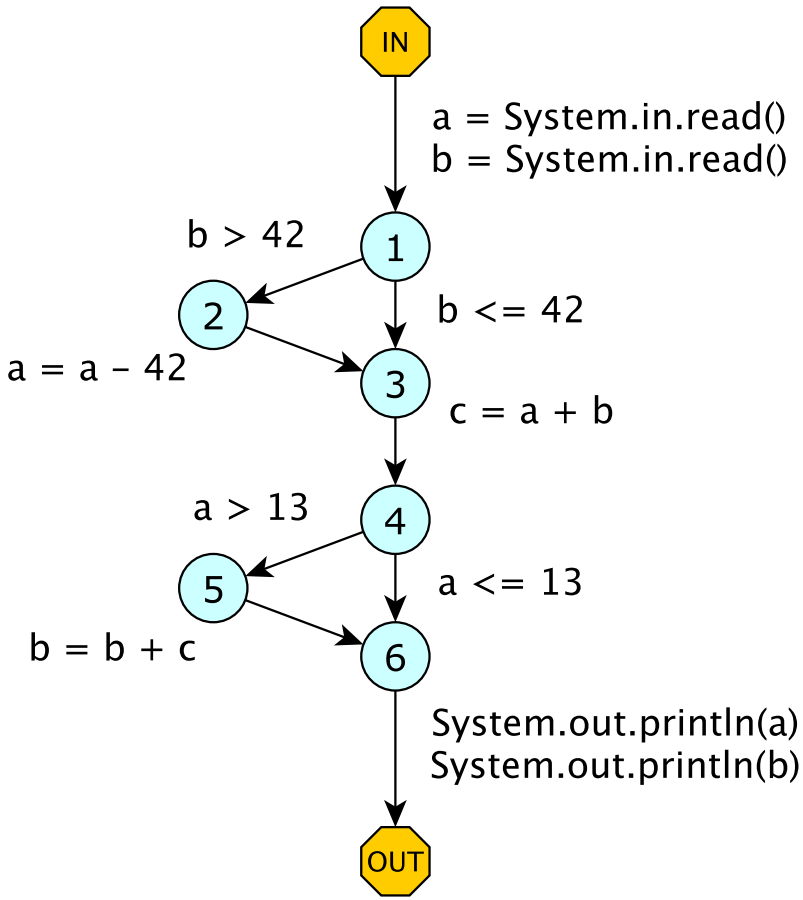

IN
a = System.in.read()
b = System.in.read()
1
b > 42
2
b <= 42
a = a - 42
3
c = a + b
4
a > 13
5
a <= 13
b = b + c
6
System.out.println(a)
System.out.println(b)
OUT

Покрытие потока данных

Какие варианты покрытий можно предложить с использованием этих понятий?

- Покрытие всех определений
  - Для каждой интересующей нас переменной v должна быть протестирована хотя бы одна *Def-Use Chain* от **каждого** определения v до хотя бы одного использования v

All-Def Coverage



Рассмотрим следующие тесты

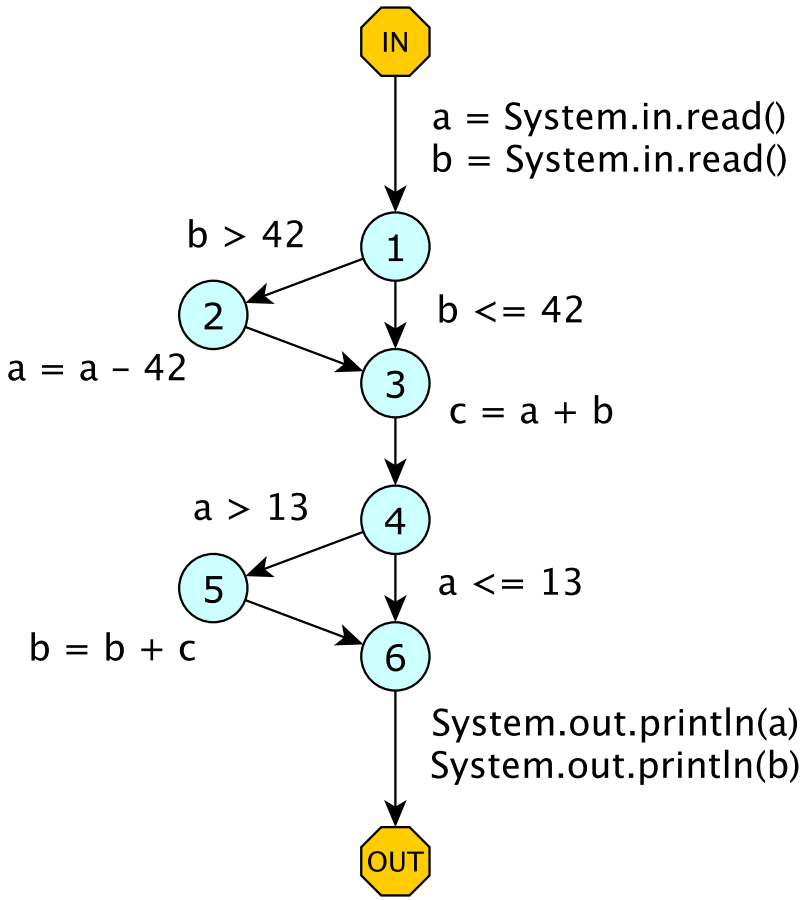
- $1 \to 2 \to 3 \to 4 \to 5 \to 6$

Покрытие потока данных

- Покрытие всех использований
  - Для каждой интересующей нас переменной v должна быть протестирована хотя бы одна *Def-Use Chain* от **каждого** определения v до **каждого** использования v

Является ли All-Use более сильным критерием по сравнению с All-Def?

All-Use Coverage



Рассмотрим следующие тесты

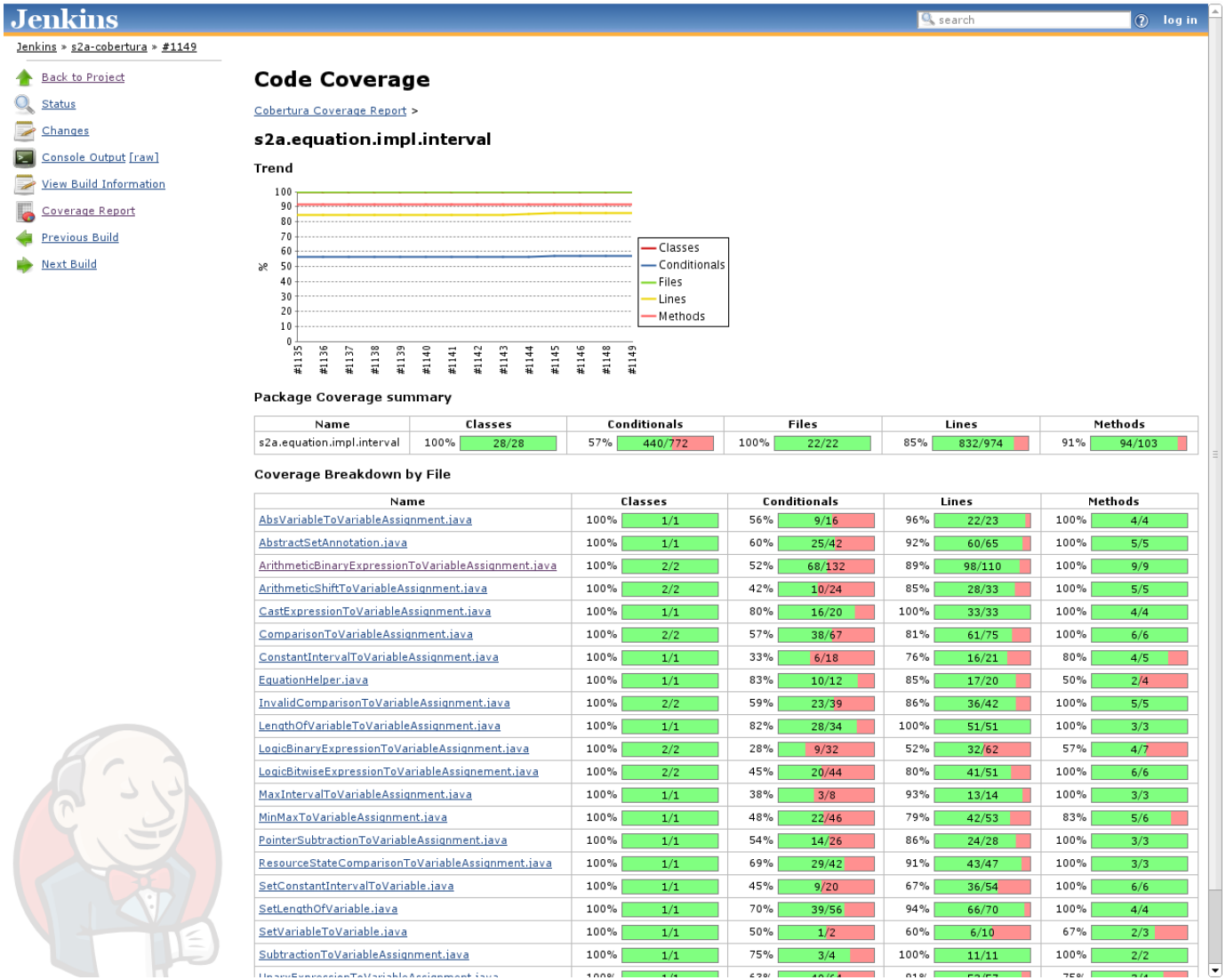
- $1 \to 2 \to 3 \to 4 \to 5 \to 6$
- $1 \to 2 \to 3 \to 4 \to 6$
- $1 \to 3 \to 4 \to 5 \to 6$

Покрытие потока данных

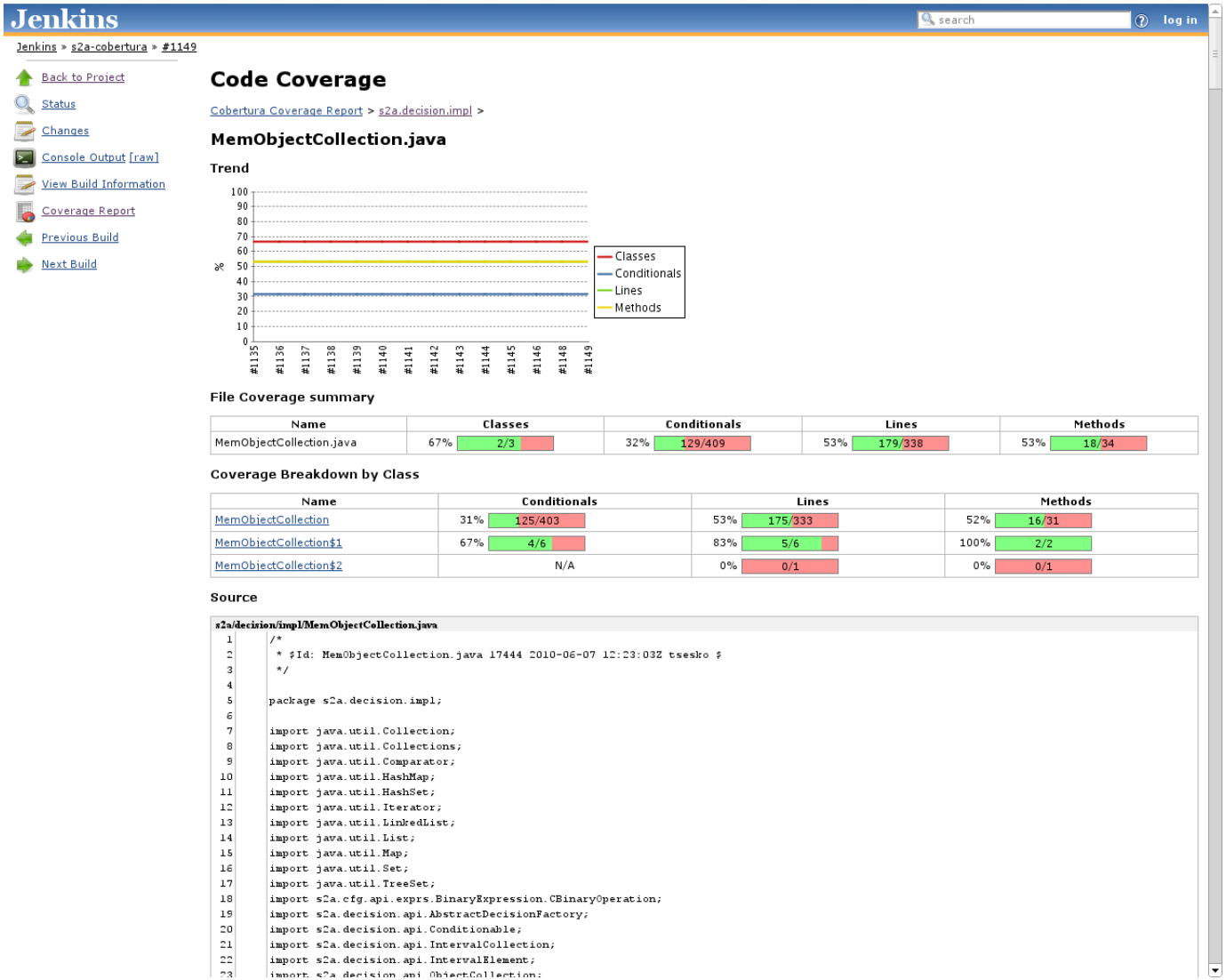
- Покрытие всех Def-Use Chain
  - Для каждой интересующей нас переменной v должны быть протестированы все возможные *Def-Use Chain* от **каждого** определения v до **каждого** использования v

Как соотносится All-Def-Use-Chain с покрытием всех путей программы?

Пример


Jenkins
search
log in
Jenkins » s2a-cobertura » #1149
Back to Project
Status
Changes
Console Output [raw]
View Build Information
Coverage Report
Previous Build
Next Build
Code Coverage
Cobertura Coverage Report >
s2a.equation.impl.interval
Trend
Classes
Conditionals
Files
Lines
Methods
Package Coverage summary
Coverage Breakdown by File

Пример


Jenkins
search
log in
Jenkins » s2a-cobertura » #1149
Back to Project
Status
Changes
Console Output [raw]
View Build Information
Coverage Report
Previous Build
Next Build
Code Coverage
Cobertura Coverage Report > s2a.decision.impl >
MemObjectCollection.java
Trend
Classes
Conditionals
Lines
Methods
File Coverage summary
Name | Classes | Conditionals | Lines | Methods
MemObjectCollection.java | 67% 2/3 | 32% 129/409 | 53% 179/338 | 53% 18/34
Coverage Breakdown by Class
Name | Conditionals | Lines | Methods
MemObjectCollection | 31% 125/403 | 53% 175/333 | 52% 16/31
MemObjectCollection$1 | 67% 4/6 | 83% 5/6 | 100% 2/2
MemObjectCollection$2 | N/A | 0% 0/1 | 0% 0/1
Source
s2a/decision/impl/MemObjectCollection.java
/*
* $Id: MemObjectCollection.java 17444 2010-06-07 12:23:03Z tsesko $
*/
package s2a.decision.impl;
import java.util.Collection;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.TreeSet;
import s2a.cfg.api.exprs.BinaryExpression.CBinaryOperation;
import s2a.decision.api.AbstractDecisionFactory;
import s2a.decision.api.Conditionable;
import s2a.decision.api.IntervalCollection;
import s2a.decision.api.IntervalElement;

Пример

```
53 1984     }
54
55          @Override
56          public String toString() {
57    96        final StringBuilder sb = new StringBuilder();
58
59    96        final List<ObjectElement> data = new LinkedList<ObjectElement>(this);
60   309        Collections.sort(data, new Comparator<ObjectElement>() {
61                  @Override
62                  public int compare(ObjectElement o1, ObjectElement o2) {
63   117                if (o1.isOrdinary() && o2.isOrdinary()) {
64   117                    if (o1.getObject() == o2.getObject()) {
65    32                        return o1.getShift() - o2.getShift();
66                          } else {
67    85                        return o1.getObject().toString()
68                                      .compareTo(o2.getObject().toString());
69                          }
70                      } else {
71     0                    return o1.toString().compareTo(o2.toString());
72                      }
73                  }
74              });
75
76    96        final Iterator<ObjectElement> iterator = data.iterator();
77   256        while (iterator.hasNext()) {
78   160            final ObjectElement e = iterator.next();
79   160            sb.append(e);
80   160            if (iterator.hasNext()) {
81    96                sb.append(", ");
82              }
83   160        }
84
85    96        return sb.toString();
86          }
87
88          @Override
89          public boolean equals(final Object o) {
90   224        if (o == this) {
91    32            return true;
92              }
93
94   192        if (!(o instanceof Collection)) {
95     0            return false;
96              }
97
98              @SuppressWarnings("unchecked")
99   192        final Collection<ObjectElement> c = (Collection<ObjectElement>) o;
100
101  192        if (c.size() != size()) {
102   64            return false;
103             }
104
105             try {
106  128            return containsAll(c);
107    0        } catch (ClassCastException unused) {
108    0            return false;
109    0        } catch (NullPointerException unused) {
110    0            return false;
111             }
112         }
113
114         @Override
115         public int hashCode() {
116    0        return super.hashCode();
```

Пример

```
493         }
494
495         public IntervalCollection minus(
496                 final ObjectCollection other) {
497     0       assert other != null;
498             assert !this.containsNonInit() && !other.containsNonInit():
499     0           "???? ?????????????????????? ???????????????? ?????? ????????????????????????????????????????????????? ??????????????????";
500
501             // ?????? ???????? ????????????????????? ???????????????? ????????????????????? ?????????????????????????? ???????????????????
502     0       if (contains(o_invalid) || other.contains(o_invalid)) {
503     0           return factory.singleton(i_max);
504             }
505
506             // ???????????? ??????????????????????
507
508     0       final IntervalCollection result = factory.createIntervalCollection();
509
510     0       for (ObjectElement ii : this) {
511     0           for (ObjectElement jj : other) {
512     0               if (ii.getObject().equals(jj.getObject())) {
513     0                   final int shift = ii.getShift() - jj.getShift();
514     0                   result.add(factory.createInterval(shift, shift));
515     0               } else {
516                         // ???????? ???????????????? ????????????????????? ???? ???????????? ?????????????
517     0                   return factory.singleton(i_max);
518                     }
519                 }
520             }
521
522     0       return result;
523         }
524
525         public IntervalCollection compare(final ObjectCollection other) {
526     0       boolean maximumIsNeeded = false;
527     0       boolean zeroIsNeeded = false;
528
529     0       final IntervalCollection result = factory.createIntervalCollection();
530
531             // ?????? ???????? ????????????????????? ???????????????? ????????????????????? ?????????????????????????? ???????????????????
532     0       if (contains(o_invalid) || other.contains(o_invalid)) {
533     0           maximumIsNeeded = true;
534             }
535
536             // ???????????? ??????????????????????
537     0       for (ObjectElement ii : this) {
538     0           if (ii.isSingle()) {
539     0               for (ObjectElement jj : other) {
540     0                   if (jj.isSingle()) {
541     0                       if (ii.getObject().equals(jj.getObject())) {
542     0                           zeroIsNeeded = true;
543                             } else {
544                                 // ???????? ????????????????????? ????????????????????? ???? ???????????? ?????????????
545     0                           maximumIsNeeded = true;
546                             }
547                         }
548                     }
549                 }
550             }
551
552     0       if (maximumIsNeeded) {
553     0           result.add(i_max_point);
554             }
555
556     0       if (zeroIsNeeded) {
```

Другой способ оценки полноты

Какими способами можно управлять выполнением кода?

- Изменением входных данных
- Изменением самого исходного кода

Как можно оценить полноту тестирования, изменяя исходный код?

Идеальный тест

- Идеальный тест работает только на тестируемой программе
- При любом изменений он перестает проходить

В этом заключается основная идея мутационного тестирования

Мутационное тестирование

- Исходная программа подвергается мутации, в результате получается набор из N мутантов
- После этого имеющиеся тесты запускаются на этих мутантах
- Если тест не проходит на мутанте, то говорят, что тест «убивает» этого мутанта
- Доля «убитых» мутантов показывает, насколько полно данный набор тестов покрывает нашу программу

Как сделать мутанта?

- Что можно сделать с исходным кодом?
  - Добавить новый код
  - Удалить старый код
  - Изменить существующий код

Набор синтаксических трансформаций, изменяющих исходный код

Как сделать мутанта?

```
1 int sumCollection(final @NotNull Collection<Integer> c) {
2     int sum = 0;
3     for (int i : c) {
4         sum += i;
5     }
6     return sum;
7 }
```

Как можно мутировать данный фрагмент кода?

Как сделать мутанта?

- Некоторые мутанты будут синтаксически некорректны
- Другие мутанты будут семантически некорректны
- Оставшиеся мутанты подходят для использования в мутационном тестировании

Какие проблемы есть у мутационного тестирования?

Недостатки мутационного тестирования

- Данный вид тестирования практически невозможно выполнять вручную
  - Количество необходимых для оценки покрытия мутантов пропорционально объему анализируемого ПО
  - Даже для небольшой программы получение достаточного числа мутантов вручную является практически невозможным
- Сильно возрастают затраты на проведение тестирования
  - Вместо одного запуска каждого теста требуется выполнить N запусков
  - Кроме того, дополнительное время тратится на генерацию мутантов

Оценка тестового покрытия

Мы же уже поговорили об оценке тестового покрытия?

Оценка тестового покрытия **программы**

Оценка **самого** тестового покрытия

Who observes the observer? ©

Оценка тестового покрытия

Какая проблема есть у покрытия потока управления?

Чувствительность к изменениям в исходном коде

Оценка тестового покрытия

Какая проблема есть у покрытия потока управления?

Чувствительность к изменениям в исходном коде

Нестабильность тестового покрытия

```
1 ...
2 if (a && b && c) {
3     ...
4 }
```

```
1 bool cond = a && b && c;
2 if (cond) {
3     ...
4 }
```

Что будет с покрытием MC/DC?

Нестабильность тестового покрытия

- Чем сильнее влияют изменения в программе на тестовое покрытие, тем более нестабильным (fragile) является тестовое покрытие
- Чем более нестабильным является тестовое покрытие, тем менее адекватно оно оценивает качество тестирования

Что же делать?

Оценка тестового покрытия

The whole is more than the sum of its parts ©

- Можно применить мутационное тестирование для оценки тестового покрытия
  - Ограничить набор трансформаций
  - Посмотреть на изменение тестового покрытия для мутантов
  - ...
  - PROFIT!

Оценка тестового покрытия

Какая проблема есть у мутационного тестирования?

Эквивалентные мутанты

Оценка тестового покрытия

Какая проблема есть у мутационного тестирования?

Эквивалентные мутанты

Эквивалентные мутанты

```
1 int sumCollection(final @NotNull Collection<Integer> c) {
2     int sum = 0;
3     for (int i : c) {
4         sum += i;
5     }
6     return sum;
7 }
```

Почему эквивалентные мутанты – это плохо?

Эквивалентные мутанты

- Эквивалентные мутанты «зашумляют» итоговую оценку качества тестирования
- Из-за шума снижается адекватность оценки

Что же делать?

Оценка тестового покрытия

The whole is more than the sum of its parts ©

- Можно применить тестовое покрытие для оценки мутационного тестирования
  - Для каждого мутанта записывается трасса его выполнения
  - У эквивалентных мутантов будут похожие трассы выполнения
  - ...
  - PROFIT!

Содержание

Тестовый оракул

Magic 8-ball тестирования

Тестовый оракул

- В чем заключается проблема тестового оракула?

Его нет!

Тестовый оракул

- Вид тестового оракула очень сильно зависит от того, какую эталонную модель мы используем

- kd-tree
- stoi
- md5sum
- PDF reader

Виды тестовых оракулов

Точность

vs

Полнота

Слабые оракулы

- Падение
  - Segmentation fault
  - Core dump
- Сбой при работе в обычном окружении
  - NullPointerException
- Сбой при работе в специальном тестовом окружении
  - Valgrind

- Хорошая точность
- Плохая полнота

Средние оракулы

- Assertions
- Тесты

- Хорошая точность
- Средняя полнота

Сильные оракулы

- Эталонная реализация
- «Обратная функция»

- Средняя точность
- Хорошая полнота

Генерация оракулов

Можно ли генерировать оракула автоматически?

Да!

- Слабые оракулы
- Средние оракулы

Генерация оракулов

Можно ли генерировать оракула автоматически?

Да!

- Слабые оракулы
- Средние оракулы

Генерация слабых оракулов

- Все уже есть
  - Слабый оракул предоставляется средой выполнения
  - Если что-то упало, мы всегда об этом узнаем
- В явном виде используются весьма редко
  - Случайное тестирование
  - ...

Генерация средних оракулов

- Assertions
- Тесты

- В большинстве случаев разрабатываются вручную

Как можно сгенерировать их автоматически?

Генерация assertions

- Assertions проверяют корректность внутреннего состояния
- Как автоматически сгенерировать assertions?

Machine learning

Генерация assertions

- Собираем информацию о выполнении программы
- Извлекаем domain-specific design из исходного кода
- Выводим закономерности в работе программы
- ...
- PROFIT!

- Хорошо работает для FSM-подобных программ
- Плохо – для всех остальных

Генерация тестов

- Тесты – один из видов эталонной модели поведения
- Как автоматически сгенерировать эталонную модель?

Мутационное тестирование

Генерация тестов

Вспомним, что:

Идеальный тест работает только на тестируемой программе

- Если тест проходит на мутанте, это плохо
- Собираем информацию о выполнении оригинальной программы и мутанта
- Анализируем разницу
- ...
- PROFIT!

W.I.L.T.

Содержание

Homework

Предложите несколько вариантов тестовых оракулов для следующих программ

- clang++
- pdflatex
- libFFT
- Chromium