Статический анализ кода в IDE

Андрей Власовских
разработчик PyCharm в JetBrains
2012-03-28

Статический анализ

- Анализ кода без его исполнения
  - Динамика: `fact(20)` ?
  - Статика: `fact(x) >= 0` ?

```
public static int fact(int x) {
  int acc = 1;
  for (int i = 1; i <= x; i++) {
    acc *= i;
  }
  return acc;
}
```

Области применения

- Текстовые редакторы
- Компиляторы
- IDE
- Инструменты поиска багов
- Инструменты доказательства программ

Плюсы и минусы IDE

- Плюсы
  - Интегрируются со средствами сборки, отладки, тестирования, версионирования и т. д.
  - Редактирование и навигация, «знающие» структуру кода
- Минусы
  - Требовательны к памяти и CPU
  - Делают сразу много вещей, взаимодействуют только с тем, что предусмотрено

IDE как средство анализа кода

- Помощь в редактировании
  - Дополнение кода
  - Рефакторинги: переименование, перемещение
- Изучение кода
  - Поиск и навигация по коду, извлечение связей
- Поиск дефектов в коде
  - Подсветка ошибок и предупреждений
  - Предложение по улучшению кода

Сравнение средств статического анализа

- Введение
- Сравнение средств статического анализа
- Статический анализ в IDE на примерах

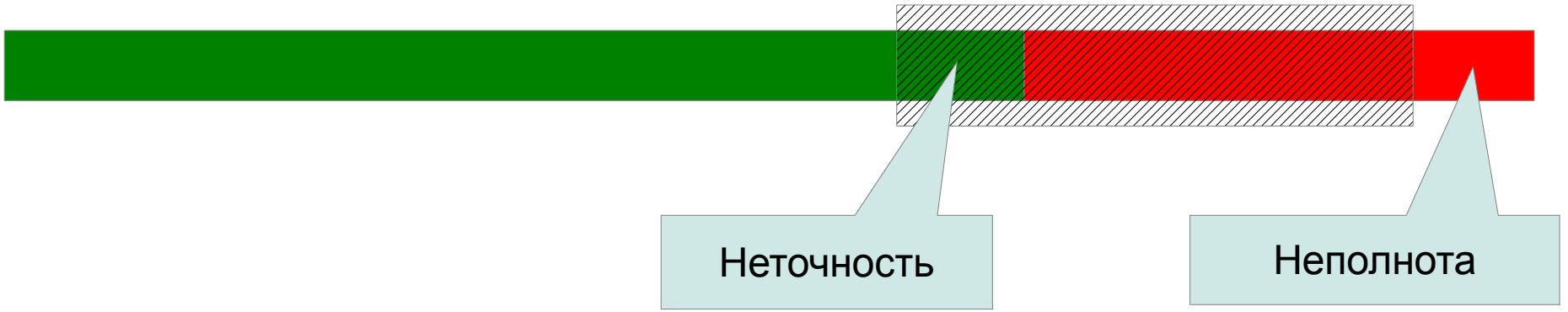
В любой программе есть ошибки

- Даже в простой
  - fact(20) = -2102132736 !?
- Места в программе, в которых есть ошибка на каком-либо из наборов данных:

```
public static int fact(int x) {
  int acc = 1;
  for (int i = 1; i <= x; i++) {
    acc *= i;
  }
  return acc;
}
```

Точность и полнота анализа

- Точность
  - Найденные при анализе ошибки на самом деле являются ошибками
- Полнота
  - Все настоящие ошибки находятся в ходе анализа



Текстовые редакторы

- Подсветка
- Ошибки в лексемах
- Подсказки без контекста

```
public static int fact(int x) {
  int acc = 1u32;
  for (int i = 1; i <= x; i++) {
    acc *= i;
  }
  return acc;
}

i<caret> // int, interface, ...
```

Текстовые редакторы (2)

- Сверхбыстрый анализ
  - O(1) от числа файлов
  - В реальном времени
- Очень узкий класс лексических ошибок
- Точный и полный

Компиляторы

- Ошибки синтаксиса
- Ошибки области видимости, типизации

```
public static int fact(x: int) {
  final int acc = "foo";
  for (i = 1; i <= x; i++) {
    acc *= i;
  }
  return acc;
}
```

Компиляторы (2)

- Медленный анализ
  - O(N) от числа файлов, +инкрементальность
  - В обозримом времени (compiling!)
- Довольно узкий класс ошибок
- Точный и полный

IDE

- Инспекции: дополнительные ошибки
- Предложения по улучшению
- Контекстные подсказки
- Рефакторинги

```
// static?, final?, @NotNull?
public int fact(int x, Log log) {
  if (x < 0) {
    log.warn("x must be positive");
  }
  int acc = 1;
  for (int i = 1; i <= x; i++) {
    acc *= i;
  }
  return <caret> // x, acc
}
...
fact(20, null);
```

IDE (2)

- Быстрый анализ
  - O(1) от числа файлов, но индексация O(N)
  - В реальном времени
- Более широкий класс ошибок и предложений
- Высокая точность, средняя полнота

Инструменты поиска багов

- Возможные ошибки
- Сложно вычисляемые ошибки

```
public static int fact(int x) {
  int acc = 1;
  for (int i = 1; i <= x; i++) {
    acc *= i;
  }
  return acc;
}
```

-2^31 < x < 2^31 => 0 < i < 2^31 => acc переполнится

но если forall fact(x): -2^31 < x < **13** => ок

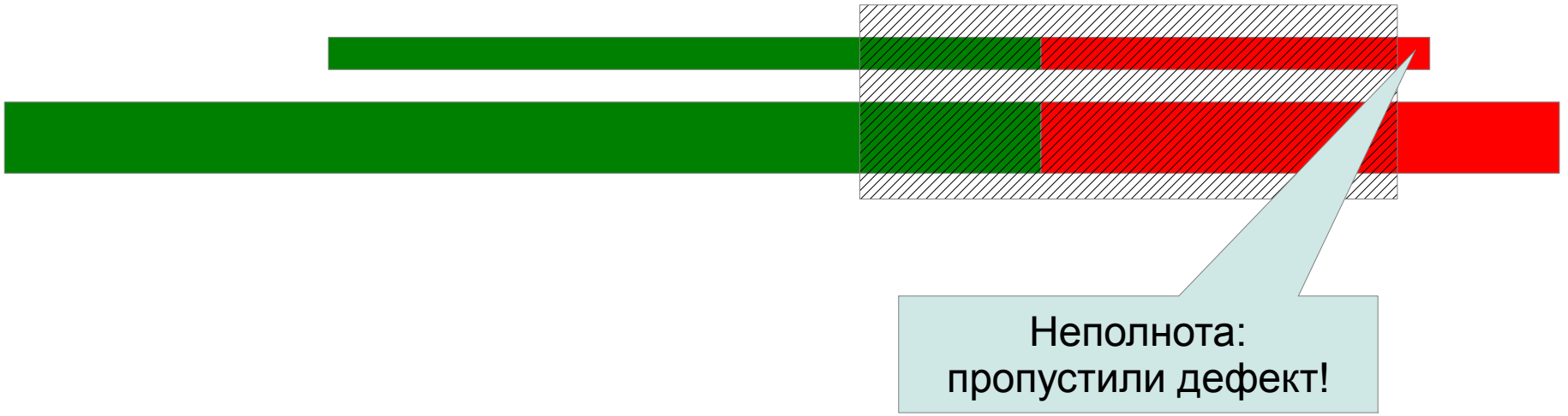
Инструменты поиска багов (2)

- Пример: выход за границы списка

```
public interface List<T> {
  int size();
  // 0 <= i < size()
  T get(int i);
}
...
void test(List<T> xs, int i) {
  xs.get(i);
  if (i < xs.size() && i >= 0) {
    xs.get(i);
  }
}
```

Инструменты поиска багов (3)

- Очень медленный анализ
  - O(N) от числа файлов
  - Много вычислений и ресурсов
- Широкий класс ошибок и предложений
- Средняя точность, высокая полнота



Инструменты доказательства программ

- Док-во свойств программ с помощью инструмента (proof assistant)

```
public static int fact(int x) {
  int acc = 1;
  for (int i = 1; i <= x; i++) {
    acc *= i;
  }
  return acc;
}
```

пусть переполнения нет, тогда:

forall x: fact(x) => 0
forall x: fact(x) => x
forall x: fact(x) = P(i) for i from 1 to x

аксиома: forall x: если x > 2^31 => переполнение

=> fact(x) равен факториалу x, если факториал x меньше 2^31 (это верно если x меньше **13**)

Инструменты доказательства программ (2)

- Страшно медленный анализ
  - Частично ручной (умственный) труд!
- Возможность анализа всех ошибок
- Точность, полнота

Тестирование

- Динамический анализ
- Ошибки почти любого класса
  - Влключая алгоритмические
- Точность, никакой полноты
  - Доказательство *наличия* ошибки в коде

```
@Test
public void fact20IsPositive() {
  assertTrue(fact(20) > 0);
}
```

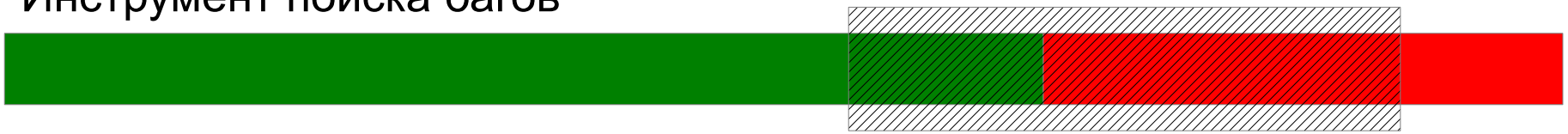
Сравнение полноты и точности

Текстовый редактор

Компилятор

IDE

Инструмент поиска багов



Инструмент доказательства

Тестирование (динамический анализ)

Статический анализ в IDE на примерах

- Введение
- Сравнение средств статического анализа
- Статический анализ в IDE на примерах

Неинициализированная переменная

- Использование до инициализации
- Шаги
  - Лексический разбор
  - Синтаксический разбор
  - Обход синтакс. дерева
  - Проверка декларации переменной
  - Построение графа потока управления
  - Поиск инициализаций по графу

```
public static int fact(int x) {
  int i;
  final int acc = 1;
  while (i <= x) {
    acc = acc * i;
    i = i + 1;
  }
  return acc;
}
```

Лексический разбор

- Текст программы в список токенов
- Регулярные выражения

```
public static int fact(int x) {
  final int acc = 1;
  while (i <= x) {
    acc = acc * i;
    i = i + 1;
  }
  int i = 1;
  return acc;
}
```

```
KEYWORD "public" 0,6
WHIESPACE " " 6,1
KEYWORD "static" 7,6
WHITESPACE " " 13,1
...
RPAREN ")"
WHITESPACE " "
LBRACE "{"
WHITESPACE "\n  "
KEYWORD "final"
...
```

Синтаксический анализ: AST

- Список токенов в синтаксическое дерево (AST)
- Контекстно-свободные грамматики, BNF
- Методы парсинга: LR, LL(*), LALR, GLR

```
KEYWORD "public"
WHIESPACE " "
KEYWORD "static"
WHITESPACE " "
...
RPAREN ")"
WHITESPACE " "
LBRACE "{"
WHITESPACE "\n  "
KEYWORD "final"
...
```

```
TreeNode
  TreeNode
    LeafNode "public"
    LeafNode "static"
  TreeNode
    LeafNode "int"
  TreeNode
    LeafNode "fact"
    TreeNode
      LeafNode "("
      TreeNode
        TreeNode
          LeafNode "int"
        LeafNode "i"
      LeafNode ")"
  TreeNode
    ...
```

Синтаксический анализ: PSI

- Дерево AST в дерево структуры программы PSI
  - `FunctionDefinition.getName(), FunctionDefition.getParameters()`

```
TreeNode
  TreeNode
    LeafNode "public"
    LeafNode "static"
  TreeNode
    LeafNode "int"
  TreeNode
    LeafNode "fact"
    TreeNode
      LeafNode "("
      TreeNode
        TreeNode
          LeafNode "int"
        LeafNode "i"
      LeafNode ")"
  TreeNode
    ...
```

```
FunctionDefinition
  FunctionModifiers
    LeafNode "public"
    LeafNode "static"
  TypeDecl
    LeafNode "int"
  FunctionPrototype
    LeafNode "fact"
    FunctionParameters
      LeafNode "("
      ParameterDecl
        TypeDecl
          LeafNode "int"
        LeafNode "i"
      LeafNode ")"
  FunctionBody
    ...
```

Инспекции кода

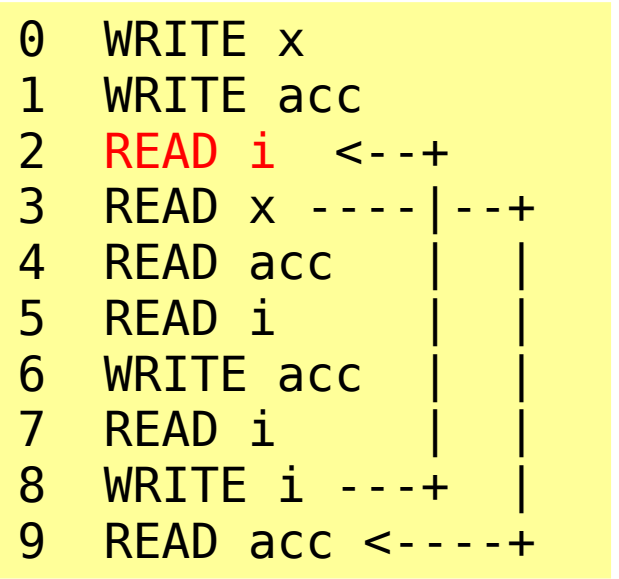
- Реализуют шаблон Visitor для элементов дерева PSI
- Сообщают о проблемах с элементами дерева

```
public class UninitializedVisitor extends PsiVisitor {
  public void visitVariableReference(VariableReference var) {
    // Найти область определения var
    // Проверить, что var определена
    // Если var локальная, построить граф потока управления
    // Найти инициализации по графу
  }
}
```

Граф потока управления (CFG)

- Пути выполнения программы с учётом ветвлений, циклов, исключений и т. д.

```
public static int fact(int x) {
  int i;
  final int acc = 1;
  while (i <= x) {
    acc = acc * i;
    i = i + 1;
  }
  return acc;
}
```



Переименование метода

- Найти и обновить все ссылки на метод
- Шаги
  - Поиск ссылок на метод в индексе
  - Проверка найденных ссылок
  - Переименование найденных ссылок

```
package other;
import libfact1.Utils;

public class fact {
  ...
  Utils.fact(42);
```

```
package myfact;

public static int fact(int x) {
  int acc = 1;
  for (int i = 1; i <= x; i++) {
    acc *= i;
  }
  return acc;
}
```

```
static import myfact.fact;

@Test
public void fact20IsPositive() {
  assertTrue(fact(20) > 0);
}
```

Индексы и заглушки PSI

- Быстрый поиск в файлах без их чтения и разбора
- Заглушки (PSI stubs): ссылки на закрытые файлы
  - Дерево только внешне видимых элементов PSI в бинарном формате
  - Прозрачное переключение между stubs и AST
  - Индексы могут ссылаться на элементы заглушек

IntelliJ IDEA Community Edition

- Open Source: Apache License
  - https://github.com/JetBrains/intellij-community
- Свои плагины
  - Инспекции, рефакторинги, языки и фреймворки
  - http://confluence.jetbrains.net/display/IDEADEV/
- Новый язык Kotlin для JVM от JetBrains
  - http://www.jetbrains.com/kotlin/

http://pirx.ru/

@vlasovskikh