Исследование и разработка метода редукции программ на языке Kotlin

Даниил Степанов

5 апреля 2019 г.

Санкт-Петербургский политехнический университет Петра Великого

Этапы исправления ошибки в программном коде

- Воспроизведение
- **Локализация**
- Изучение
- Исправление
- Тестирование

Пример

GCC ломается на следующем файле:

```
extern int printf (const char *, ...);
static char
(safe_unary_minus_func_int8_t_s)(char si )
{
  return
    (si==(-128)) ?
    ((si)) : -si;
  ...
  2683 lines of code
```

Классический дельта-дебаггинг

Дано: $test$ и c_x, такие что $test(c_x) = fail$
Необходимо найти $c'_x = ddmin(c_x)$, такое что
$c'_x \subseteq c_x, test(c'_x) = fail$

$ddmin(c_x) = ddmin_2(c_x, 2)$ где

$$ddmin_2(c'_x, n) = \begin{cases} ddmin_2(\Delta_i, 2) & \text{if } test(\Delta_i) = fail \\ ddmin_2(\nabla_i, max(n-1, 2)) & \text{if } test(\nabla_i) = fail \\ ddmin_2(c'_x, min(|c'_x|, 2n)) & \text{if } n < |c'_x| \\ c'_x \end{cases}$$

где $\nabla_i = c'_x - \Delta_i, c'_x = \Delta_1 \cup \Delta_2 \cup ... \cup \Delta_n$, все Δ_i попарно не пересекаются

Пример

{1, 2, 3, 4, 5, 6, 6, 8}	X
{1, 2, 3, 4}	V
{5, 6, 6, 8}	X
{5, 6}	V
{6, 8}	V
{5}, {6}, {6}, {8}	V
{5, 6, 6}	X
{5, 6}	V
{6, 6}	X
{6}	V
{6}	V
{6, 6}	X

“Простой” дельта-дебаггинг

- Работает на уровне строк
- Не учитывает структуру кода

```
void f() {
    int x;
    int y;
    if (x != 0) {
        y = x;
    } else {
        return 0;
    }
    while (y != 0) {
        y--;
    }
    return y;
}
```

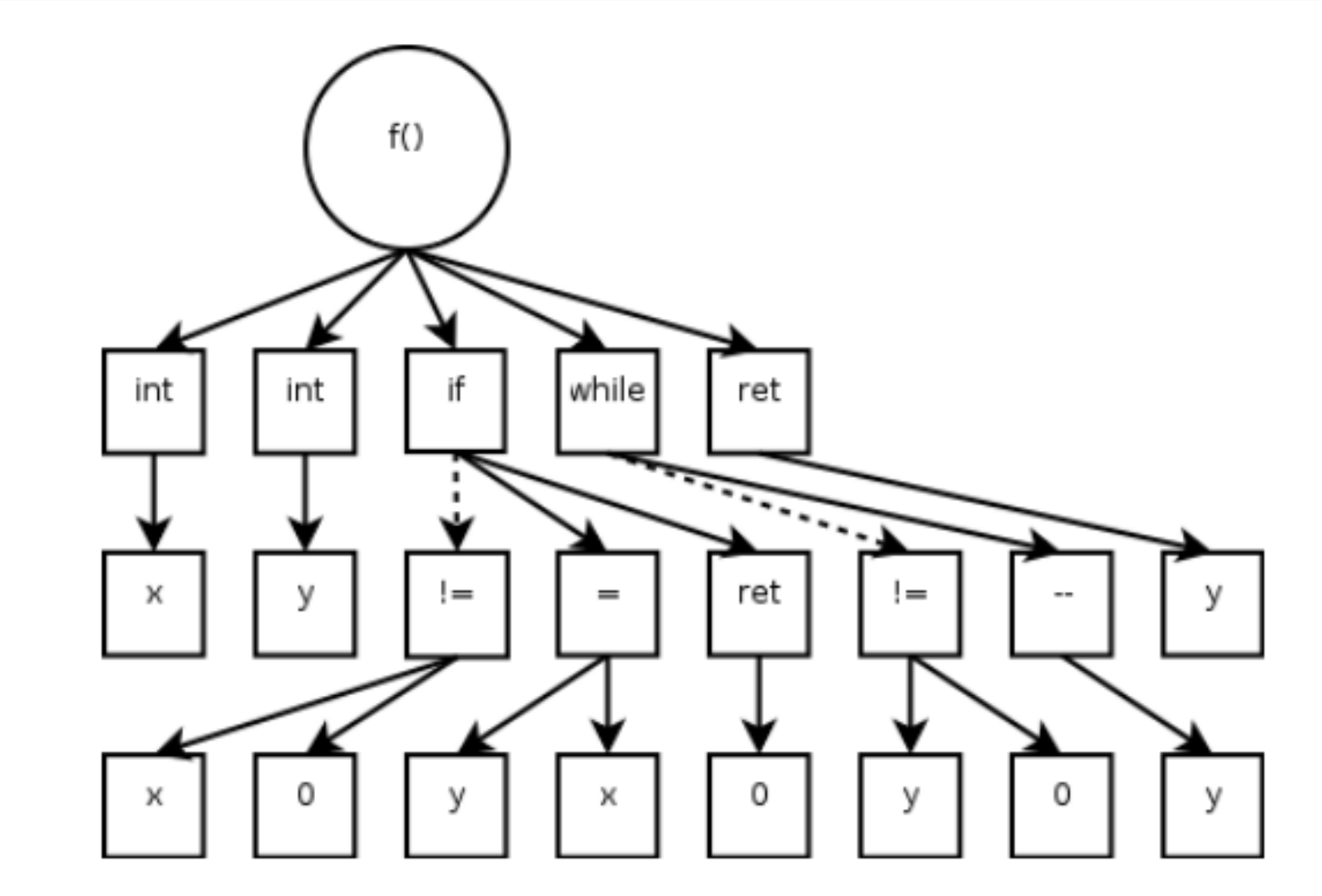
Дельта-дебаггинг + topformflat

```
//Level = 1
void f() {
     int x;
     int y;
     if (x != 0) { x *= 3; y =
          x;}
     else {return 0;}
     while (y != 0) {y--;}
     return y;
 }
```

```
//Level = 1
fun f(): Int {
     var x = 1
     var y = 0
     if (x != 0) { x *= 3 y = x }
     else { return 0 }
     while (y != 0) { --y }
     return y
}
```

Не подходит для языков, где переносы строки являются критичными

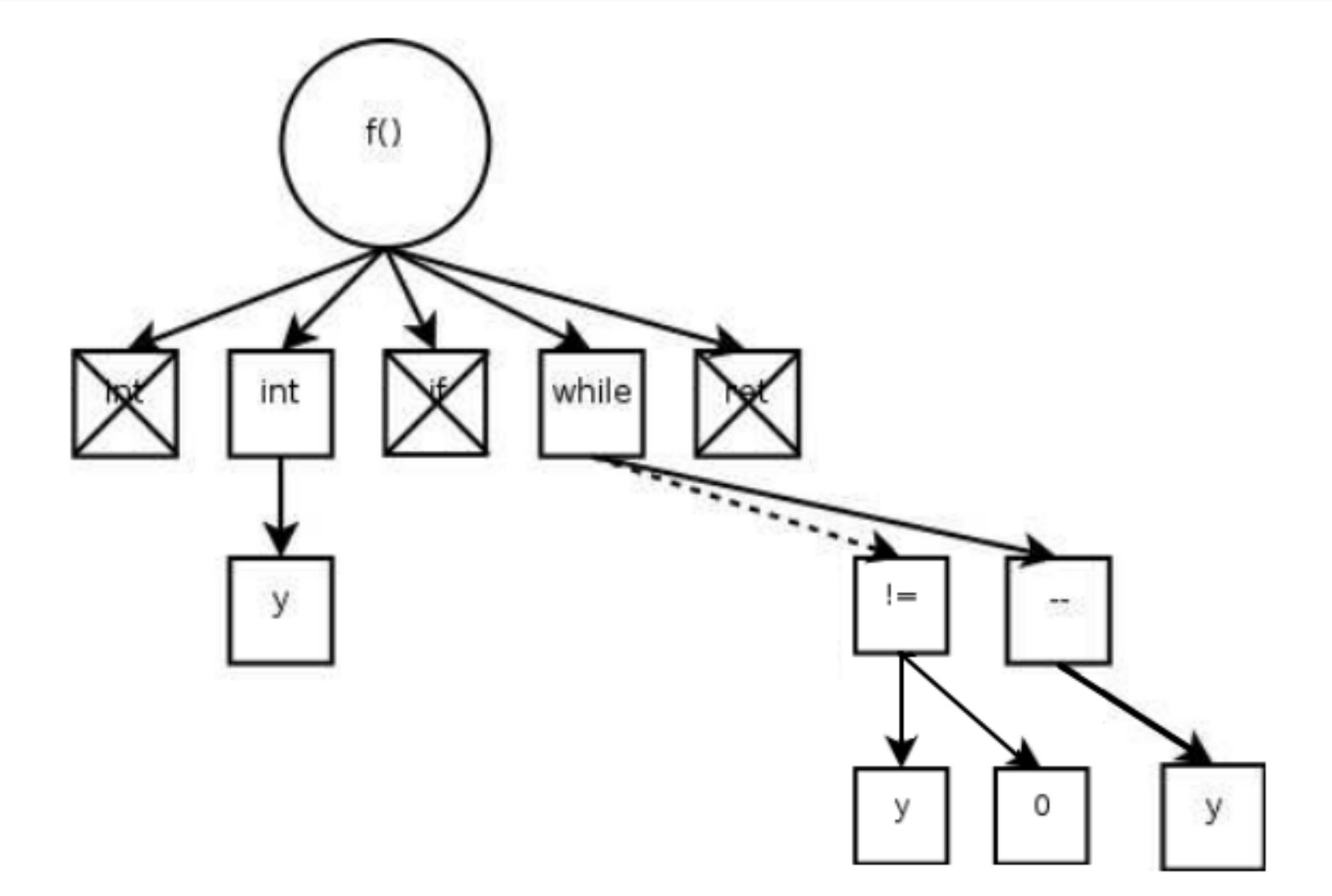
Иерархический-дельта дебаггинг



Иерархический дельта-дебаггинг

$level \leftarrow 0$
$node \leftarrow getNodesOfLevel(tree, 0)$
while($nodes \neq \emptyset$) do
    $minconfig \leftarrow DDMIN(nodes)$
    $delete(tree, level, minconfig)$
    $level \leftarrow level + 1$
    $nodes \leftarrow getNodesOfLevel(tree, level)$
endwhile

Иерархический дельта-дебаггинг



Существующие инструменты

- Picireny
  - Иерархический дельта-дебаггер
  - ANTLR v4 grammar
  - Rule
- Creduce (Delta)
  - Редуктор для языка C
  - Состоит из набора трансформаций над исходным кодом
  - 57 компиляторных багов
    - Delta: 8600 байт
    - Creduce: 120 байт

Мотивация

- Существует генератор случайных тестов для компилятора котлина
- Зачастую компиляторные баги находятся в очень больших проектах

ReduKtor

- Специфичные для Котлина peephole трансформации
  - x += y -> x = y
  - val s: String = t ?: "" -> val s: String = ""
  - while -> if
  - ...
- Иерархический дельта-дебаггинг
- Слайсинг
- Трансформации над PSI
  - Упрощение when
  - Замена тела функции на константу
  - Инлайнинг
  - Переименование переменных
  - ...

Прогресс

На данный момент реализованы:

- Некоторые peephole трансформации
- Алгоритм иерархического дельта-дебаггинга
- Система для запуска трансформаций и проверки результирующих программ
- Работает для компиляторных тестов и обычных программ (необходимые свойства поведения программы задаются с помощью скрипта)

Планы на будущее

- Больше трансформаций
- Редукция проектов целиком
- Поддержка языка Java
- Апробация на реальных багах

Контакты

stepanov@kspt.icc.spbstu.ru

ReduKtor repository:

https://bitbucket.org/vorpal-research/reduktor