

Разработка системы условных эффектов с реализацией на языке программирования Kotlin

Саввинов Дмитрий Кимович

научный руководитель: руководитель команды
компилятора языка Kotlin, С.Е. Ерохин

СПб АУ НОЦНТ РАН

12 июня 2017 г.

Работает:

```
1 fun foo(x: Any) {  
2     if (x is String) {  
3         println(x.length)  
4     }  
5 }
```

Не работает:

```
1 fun isString(x: Any) =  
2     x is String  
3  
4 fun foo(x: Any) {  
5     if (isString(x)) {  
6         x.length  
7     }  
8 }
```

Компилятор не знает, что:
`isString(x) == true ⇔ x is String`

Не работает:

```
1 fun test(x: Any) {  
2     assert(x is String)  
3     x.length  
4 }
```

Компилятор не знает, что:
assert(c) завершился

⇔

c == true

Не работает:

```
1 fun test(list: List<Any>)  
2     val ss = list.filter {  
3         x -> x is String  
4     }  
5     ss[0].length  
6 }
```

Компилятор не знает две вещи:

- Лямбда возвращает true только на String
- filter оставляет только те элементы, на которых лямбда вернула true

Предметная область. Инициализация переменных.

Работает:

```
1 val x: Int
2 x = 42
3 println(x)
```

Не работает:

```
1 val x: Int
2 // Reassignment
3 run { x = 42 }
4
5 // Not initialized
6 println(x)
```

Компилятор знает, что лямбда пишет в x, но не знает, что она выполнится **ровно раз**

Аналогичным образом в Kotlin реализованы synchronized, try-with-resources, async-await, и Т.д.

Цель: разработка системы, выполняющей анализ кода на языке программирования Kotlin посредством использования информации о поведении вызываемых функций

Требования к системе:

- Должна решать как минимум вышеописанные проблемы
- Должна быть поддерживаемой
- Должна быть расширяемой

Задачи:

- Проанализировать существующие решения
- Разработать систему
- Реализовать разработанную систему в компиляторе Kotlin
- Провести анализ полученного решения, выявить его достоинства и недостатки

1 Системы эффектов

- «An Object-Oriented Effects System». Greenhouse, Boyland. 1999
- JSR-308 (Checker Framework)
- Eff-language

2 Контракты

- Контракты в C#
- Eiffel
- Аннотация @Contract в IDEA

3 Языки спецификации

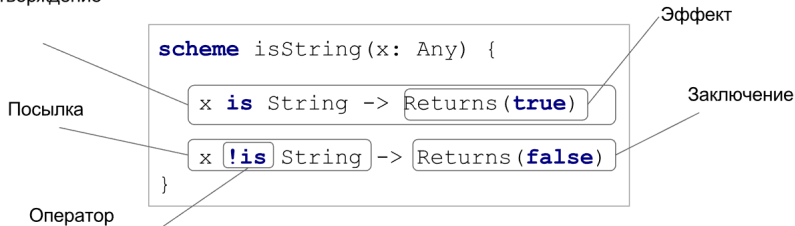
- The Z Notation
- ANSI C Specification Language
- Larch

Definition

Схема эффектов – описание эффектов вычислений и УСЛОВИЙ, вызывающих эти эффекты.

```
fun isString(x: Any): Boolean {  
    return x is String  
}
```

Утверждение



Комбинирование схем. Постановка проблемы

Даны две схемы:

```
schema isString(x: Any) {  
  x is String -> Returns true  
  x !is String -> Returns false  
}
```

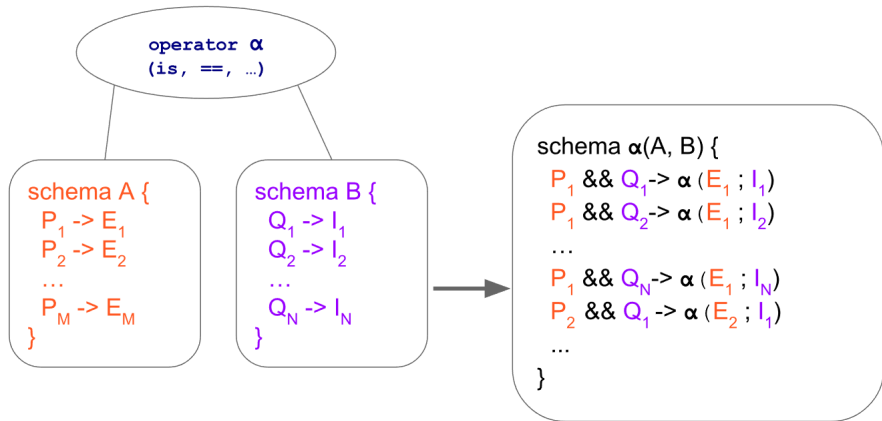
```
schema assert(c: Bool) {  
  c == true -> Returns true  
  c != true ->  
    Throws AssertionError  
}
```

Пусть выполняется **вложенный** вызов:
assert(isString(x))

Хотелось бы получить **комбинированную** схему
для этого вызова:

```
schema assert(isString(x)) {  
  x is String -> Returns true  
  x !is String -> Throws AssertionError  
}
```

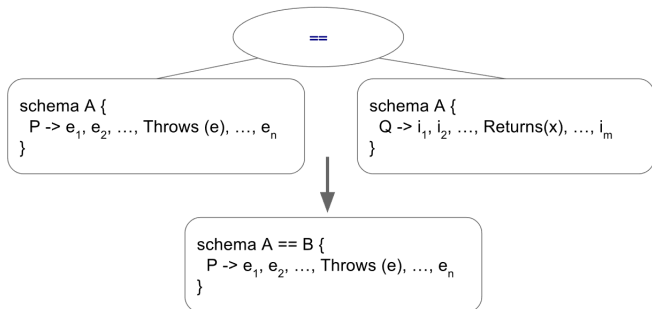
Комбинирование схем. Общее решение



- Преобразование α определяется оператором
- Как правило, определяется достаточно тривиально и *только на небольшом классе эффектов*

● $==(Returns\ x; Returns\ y) \equiv Returns(x == y)$

Комбинирование схем. Частичные вычисления



Эффекты правой части не успели выполняться!

Definition

Исход – специальный класс эффектов (Returns и Throws), которые говорят об успешности вычислений. Система эффектов «знает» про все исходы.

- Явная аннотация на функции

```
1 @Effects("""  
2     condition == true -> Returns unit;  
3     condition != true -> Throws AssertionError  
4 """)  
5 fun myAssert(condition: Boolean): Unit {  
6     if (!condition)  
7         throw AssertionError("Assertion failed")  
8 }
```

- Вывод из тела функции

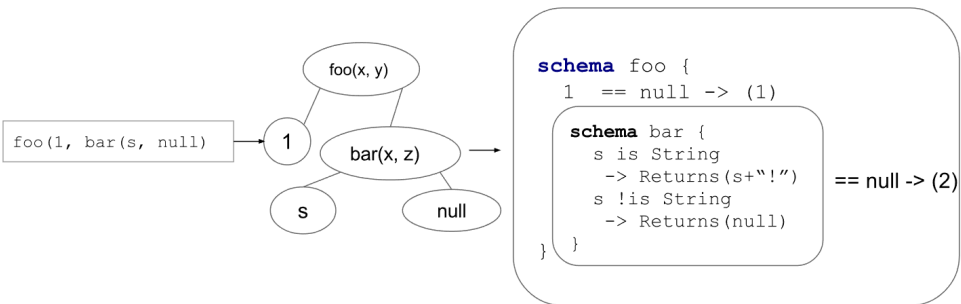
В текущей реализации поддерживаются только тела, состоящие из одного выражения (expression):

```
1 fun isString(x: Any?) = x is String
```

Реализация. Полный алгоритм

Пусть дан вызов `foo(1, bar(s, null))`.

- 1 Из IR компилятора убирается лишняя информация, проверяется, что resolving выполнен, и т.д.
- 2 Все вызовы заменяются на соответствующие схемы, выполняется подстановка переменных



Реализация. Полный алгоритм

- 4 Выполняется комбинирование схем, получаем одну большую схему, описывающую все эффекты данного вызова
- 5 Выполняется сокращение схемы: вычисляются константы, откидываются невозможные ветки вычислений и т.д.

```
schema foo {  
  1 == null -> (1)  
  schema bar {  
    s is String  
    -> Returns (s+"!")  
    s !is String  
    -> Returns (null)  
  }  
}
```

== null -> (2)

```
schema foo {  
  1 == null -> ...  
  s is String  
  -> Returns (s+"!" == null)  
  s !is String  
  -> Returns (null == null)  
}
```

Применение. Детерминированные вызовы

Вводим эффект `Calls(f, c)`: «Функция f будет вызвана c раз»

```
1 fun run(block: () -> Unit)
2   = block()
```

```
schema run {
  true -> Calls(block, 1)
}
```

Данный эффект «хороший» в том смысле, что он – классический *side-effect*.

Поэтому он легко описывается и не требует дополнительных конструкций при введении

Применение. Смарткасты в коллекциях

Для введения смарткастов в коллекциях нам понадобится два дополнительных оператора:

- $S \text{ at } Y$, где S - схема, Y - эффект

```
schema isString {  
  x is String  
    -> Returns true  
  x !is String  
    -> Returns false  
}
```

at (Returns true)

```
schema {  
  x is String ->  
    Returns true  
}
```

- $S \text{ typeOf } V$, где S - схема, V - переменная

```
schema {  
  x is String ->  
    Returns true  
}
```

typeOf

x

String

```
1 fun filter(l: List<T>, p: (x: T) -> Boolean): List<T> {  
2     ...  
3 }
```

```
schema filter {  
    true -> Returns List <  
        ( p(x) at (Returns true) ) typeOf x  
        тип переменной x, если p(x) вернул true  
    >  
}
```

- Была разработана грамматика, написан парсер с помощью ANTLR
- Были разработаны правила комбинирования схем эффектов
- Были поддержаны некоторые виды эффектов: Returns, Throws, Calls
- Были улучшен статический анализ языка Kotlin:
 - Смарткасты в условных выражениях, assert-подобных вызовах, коллекциях
 - Анализ (ре)инициализации переменных
- Система расширяема за счет добавления новых эффектов и операторов
- Поддерживается вывод эффектов для отдельных выражений (expressions)

- Внесение системы master-ветку компилятора Kotlin
- Аннотация стандартной библиотеки Kotlin
- Вывод эффектов по телу функции
- Исследование полезности более мощных понятий мат. логики (например, кванторы)
- Исследование полезности других эффектов (ввод-вывод, многопоточность)

Разработка системы условных эффектов с реализацией на языке программирования Kotlin

Саввинов Дмитрий Кимович

научный руководитель: руководитель команды
компилятора языка Kotlin, С.Е. Ерохин

СПб АУ НОЦНТ РАН

12 июня 2017 г.

Definition

Эффект вычисления – некоторая информация о состоянии окружения, полученная в результате исполнения данного вычисления

- Запись в переменную – эффект
- И возвращение значения – тоже эффект!
- И даже если мы просто узнали тип какой-то переменной, это тоже эффект!

Извлечение информации из схемы

```
schema foo(x) {  
  x == 0    -> Returns true, Calls(f, 1)  
  x != 0 && x != null  
              -> Returns true, Calls(f, 2)  
  x == null -> Returns false  
}
```

x == 0 -> Returns **true**, Calls(f, 1)

x != 0 && x != **null** -> Returns **true**
Calls(f, 2)

x != **null**
result == **true**
calls[f] == **EXACTLY_ONCE**

x != **null**
result == **false**
calls[f] == **AT_LEAST_ONCE**

OR

x != **null**
result == **true**
calls[f] == **AT_LEAST_ONCE**

Полнота спецификации

```
1 fun fakeIsString(x: Any?) =  
2   x is String || x is Int  
3 }
```

```
schema fakeIsString(x) {  
  x is String -> Returns true  
  x !is String -> Returns false  
}
```

```
1 if (fakeIsString(x)) {  
2   // Erroneous smartcast here!  
3   x.length  
4 }
```

```
schema fakeIsString(x) {  
  x is String -> Returns true  
  x is Int -> Returns true  
  x !is String && x !is Int  
  -> Returns false  
}
```