

# Автоматическое построение ограничений в модельном языке программирования с шаблонами функций и автовыводом типов

Ю. В. Белякова

Научный руководитель: доцент, к. ф.-м. н. С. С. Михалкович

Направление 010400 — Информационные технологии

Факультет математики, механики и компьютерных наук

Южный федеральный университет

23 июня 2012 г.

# Содержание

- 1 Постановка задачи
  - Формулировка задачи
  - Обобщенное программирование
- 2 Модельный язык PollyTL
- 3 Шаблоны функций
  - Основные идеи
  - Реализация
- 4 Результаты

# Содержание

- 1 Постановка задачи
  - Формулировка задачи
  - Обобщенное программирование
- 2 Модельный язык PollyTL
- 3 Шаблоны функций
  - Основные идеи
  - Реализация
- 4 Результаты

# Цель

Реализовать механизм шаблонов функций, при котором:

- 1 на этапе компиляции шаблона проводится **максимальная проверка семантики и автоматическое построение ограничений** на параметры шаблона, что позволяет получить:
  - раннее обнаружение ошибок;
  - быстрое инстанцирование;
  - широкие возможности по написанию обобщенного кода.
- 2 На этапе инстанцирования достаточно проверить типы на соответствие ограничениям.

# Постановка задачи

Для этого необходимо:

- 1 Разработать модельный императивный язык программирования с шаблонами функций (грамматика языка).
- 2 Реализовать front-end часть компилятора (используя средства автоматической генерации лексических и синтаксических анализаторов).
- 3 Построить алгоритмы анализа шаблонов функций, построения ограничений на шаблоны, анализа ограничений, инстанцирования шаблонов.
- 4 Реализовать middle-end часть компилятора (полный семантический анализ программы, включая применение вышеуказанных алгоритмов).

# Подходы к обобщенному программированию с использованием шаблонов

## Два подхода к семантике шаблонов

- 1 «Можно все», минимальная проверка кода шаблона, полная проверка кода при инстанцировании.
- 2 «Запрещено все, что не разрешено», полная проверка кода шаблона, проверка ограничений для инстанции.

Примеры:

- Шаблоны C++
- Универсальные шаблоны .NET
- Концепты C++

# Преимущества и недостатки

## «Можно все»:

- большие возможности по написанию обобщенного кода;
- позднее обнаружение ошибок.

## «Нельзя все, что не разрешено»:

- раннее обнаружение ошибок;
- быстрое инстанцирование;
- явное описание требований (ограничений) к типам;
- ограниченные возможности требований (требования к отдельным типам).

# Преимущества и недостатки (продолжение)

## Концепты:

- раннее обнаружение ошибок;
- быстрое инстанцирование;
- явное описание требований (ограничений) к типам;
- ограничения на связанные типы;
- пока не включены в стандарт;
- нет общедоступных реализаций.



# Содержание

- 1 Постановка задачи
  - Формулировка задачи
  - Обобщенное программирование
- 2 **Модельный язык PollyTL**
- 3 Шаблоны функций
  - Основные идеи
  - Реализация
- 4 Результаты

# Основные характеристики

Императивный язык программирования со строгой статической типизацией.

Три базовых типа: `Bool`, `Int`, `Double`. Функциональные типы  $S \rightarrow T$ , где  $S$ ,  $T$  — любые возможные типы. Например, тип  $(Int \rightarrow Double) \rightarrow (Int \rightarrow (Int \rightarrow Double))$ .

Операторы: описание переменных (включая описание с автовыводом типа), присваивание, условный оператор, оператор цикла.

Функции являются значениями функционального типа. Функция без параметров имеет тип  $void \rightarrow R$ , функция без возвращаемого значения —  $S \rightarrow void$ .

# Основные характеристики

(продолжение)

Функции могут быть **перегружены**. Экземпляры перегруженной функции должны отличаться типами.

Возможно **частичное применение** функции.

Все базовые операции над типами ( $+$ , **div**,  $||$ ,  $\dots$ ) являются «вшитыми» функциями (возможно, перегруженными).

Перегрузка шаблонов невозможна.

Внутри шаблона может быть вызвана инстанция другого шаблона, конкретизированная параметрами внешнего шаблона.

# Пример

```
// or :: Bool -> (Bool -> Bool)
fun or(Bool a, Bool b)
  return a || b;
end
```

```
// or :: Bool -> (Bool -> Int)
fun or(Bool a, Bool b)
  if a || b then
    return 1;
  else
    return 0;
  fi
end
```

```
fun app[!F, X, Y](F f, X x, Y y)
  var r = f(x);
  Bool b1, b2;
  // do smth
  if or(b1, b2) then
    ;// do smth
  else
    var c = or(b1, b2);
    #print_type1(c);
    #print_type(c);
    var n = y + c;
    // do smth
  fi
  return r;
end

main
  // do smth
  #print_type(or);
  #print_type(app);
end
```

# Печать типов

Использование директивы компилятора `#print_type`  
(`#print_type1`):

```
c  ::  app/3 (исходный тип в шаблоне)
c  ::  Int
or  ::  {Bool → (Bool → Bool), Bool → (Bool → Int)}
app ::  (app/X → app/1) → (app/X → (app/Y → app/1))
```

# Содержание

- 1 Постановка задачи
  - Формулировка задачи
  - Обобщенное программирование
- 2 Модельный язык PollyTL
- 3 Шаблоны функций
  - Основные идеи
  - Реализация
- 4 Результаты

# Реконструкция типов и шаблоны функций

Идея алгоритма реконструкции типов:

анализ выражений  $\Rightarrow$  сбор ограничений  $\Rightarrow$  анализ ограничений

На её основе реализуем механизм шаблонов функций для модельного языка:

- Параметры шаблона — типовые переменные.
- Анализ операторов языка.
- Построение ограничений на типы.
- Анализ ограничений.
- Инстанцирование шаблона — проверка типов инстанции на соответствие ограничениям.

# Реконструкция типов и шаблоны функций

(продолжение)

## Типизированное лямбда-исчисление

- Ограничение  $S = T$ .
- Алгоритм вывода типов Хиндли-Милнера:
  - правила построения ограничений для терма;
  - алгоритм унификации Хиндли-Милнера.

## Модельный язык PollyTL

- 3 вида ограничений (включая  $S = T$ )
- Алгоритмы построения и анализа ограничений:
  - правила построения ограничений для операторов шаблона и вызова инстанций;
  - 6 основных алгоритмов анализа ограничений (включая алгоритм унификации).



# Виды ошибок

которые должны быть обнаружены на этапе анализа ограничений

- 1 Противоречивое использование выражений, типизированных типовой переменной.

$X = \text{Int}$  и  $X = \text{Double} \Rightarrow X$  не может быть правильно конкретизирована

- 2 Некорректный вызов перегруженной функции

$X = S \rightarrow T$ ,  $x :: X$ ,  $\text{types}(f) = \{\text{Int} \rightarrow \text{Int}, \text{Int} \rightarrow \text{Double}\} \Rightarrow$  вызов  $f(x)$  невозможен

- 3 Невозможно определить нужную версию перегруженной функции

$\text{types}(f) = \{\text{Int} \rightarrow \text{Int}, \text{Int} \rightarrow \text{Double}\} \Rightarrow$  если  $x$  больше нигде не участвует, то для  $\text{var } x = f(5)$  тип  $x$  нельзя вывести

# Результаты анализа множества ограничений

Результатом анализа является ответ на вопрос:

Существуют ли конкретизации шаблона, при которых все выражения типизируемы?

## Замечание

Шаблон функции может быть корректен:

- 1 при *любых* конкретизациях шаблона;
- 2 только при *некоторых* конкретизациях.

# Результаты этапа компиляции шаблона функции

Если существуют корректные конкретизации шаблона, то сохраняется достаточное **множество ограничений**. В противном случае получена ошибка компиляции.

Для **инстанцирования шаблона** нужно:

- сопоставить конкретные типы параметрам шаблона;
- проверить набор ограничений.

# Содержание

- 1 Постановка задачи
  - Формулировка задачи
  - Обобщенное программирование
- 2 Модельный язык PollyTL
- 3 Шаблоны функций
  - Основные идеи
  - Реализация
- 4 Результаты

# Пример правильной программы

Модельный язык Polly

Файл    Выбрать файл    Компилировать

Использовать файл

v3.polly

```

        #print_type(c);
        #print_type(c);
        var n = y + c;
        // do smth

    fi
    return r;
end

fun iter_app[F, X](F f, X x, Int N)
    var k = 1;
    var acc = f(x);
    while k <= N do
        acc = f(acc);
    endw
    return acc;
end

main
    Double -> Double f;
    Double x = 27.9;
    #print_type(iter_app(f));

    #print_type(or);
    #print_type(some_app);
end

```

Информация о типах

```

r :: some_app::1
c :: some_app::3

c :: Int
iter_app(f) :: (Double -> (Int -> Double))
or :: ((Bool -> (Bool -> Bool)), (Bool -> (Bool -> Int)))
some_app :: ((some_app::X -> some_app::1) -> (some_app::X
-> (some_app::Y -> some_app::1)))

```

Данные отладки

```

----- after sorting
1) iter_app::F == (iter_app::1 -> iter_app::3)
2) iter_app::1 == iter_app::3
3) iter_app::2 == Bool
4) iter_app::F == (iter_app::X -> iter_app::1)
5) <=(Int, Int) :: iter_app::2
----- after processing
----- Substitution
{(iter_app::F +> (iter_app::X -> iter_app::X)),
(iter_app::1 +> iter_app::X), (iter_app::2 +> Bool),
(iter_app::3 +> iter_app::X)}
END
----- CALL TEMPLATE | iter_app(f)
----- Constraints
1) (iter_app::X -> iter_app::X) == (Double -> Double)
----- after sorting
1) (iter_app::X -> iter_app::X) == (Double -> Double)
----- after processing
----- Substitution
{(iter_app::F +> (Double -> Double)), (iter_app::1 +>
Double), (iter_app::2 +> Bool), (iter_app::3 +> Double),
(iter_app::X +> Double)}
END

```

Результат компиляции

Очистить

Компиляция прошла успешно

# Пример неправильной программы

Модельный язык Polly

Файл Выбрать файл Компилировать

Использовать файл

Информация о типах

```

в3.polly
// or :: Bool -> (Bool -> Int)
fun or(Bool a, Bool b)
  if a || b then
    return 1;
  else
    return 0;
  fi
end
// or :: Int -> (Int -> Bool)
fun or(Int a, Int b)
  return (a > 0) || (b > 0);
end
// or :: Int -> (Int -> Int)
fun or(Int a, Int b)
  if or(a, b) then
    return 1;
  else
    return 0;
  fi
end
fun bad[!T](T x, T y)
  var z = or(x, y);
  //z = z + 1;
end
  
```

Данные отладки

```

DEF TEMPLATE | bad
----- Constraints
1) or(bad::T, bad::T) :: bad::1
----- after sorting
1) or(bad::T, bad::T) :: bad::1
----- after processing
1) (bad::T, bad::1) from { (Bool, Bool); (Bool, Int);
(Int, Bool); (Int, Int) }
----- Substitution
{}
END
  
```

Результат компиляции

Обнаружены ошибки компиляции: Семантическая ошибка в файле "polly.file" Ни при каких значениях типовых переменных нельзя вывести все типы шаблона bad Позиция [ начало: (26, 1) конец (29, 4) ]

# Результаты работы

- 1 Разработан модельный язык программирования с шаблонами функций.
- 2 **Разработан механизм шаблонов**, основанный на автоматическом построении ограничений, с ранним обнаружением ошибок компиляции и быстрым инстанцированием.
- 3 **Построены алгоритмы** анализа шаблона, построения и анализа ограничений.
- 4 **Реализованы front-end и middle-end** части компилятора модельного языка, включая рассмотренный механизм шаблонов.