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

Содержание

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 обощенного 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

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 $(\texttt{Int} \rightarrow \texttt{Double}) \rightarrow (\texttt{Int} \rightarrow (\texttt{Int} \rightarrow \texttt{Double}))$.

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**, $||$, ...) 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

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

Модельный язык PollyTL

- 3 вида ограничений (включая $\mathtt{S} = \mathtt{T}$)
- 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.

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

2. Некорректный вызов перегруженной функции

   $\texttt{X} = \texttt{S} \to \texttt{T}$, $\texttt{x} :: \texttt{X}$, $\mathit{types}(\texttt{f}) = \{\texttt{Int} \to \texttt{Int}, \texttt{Int} \to \texttt{Double}\} \Rightarrow$ вызов `f(x)` невозможен

3. Невозможно определить нужную версию перегруженной функции

   $\mathit{types}(\texttt{f}) = \{\texttt{Int} \to \texttt{Int}, \texttt{Int} \to \texttt{Double}\} \Rightarrow$ если `x` больше нигде не участвует, то для `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

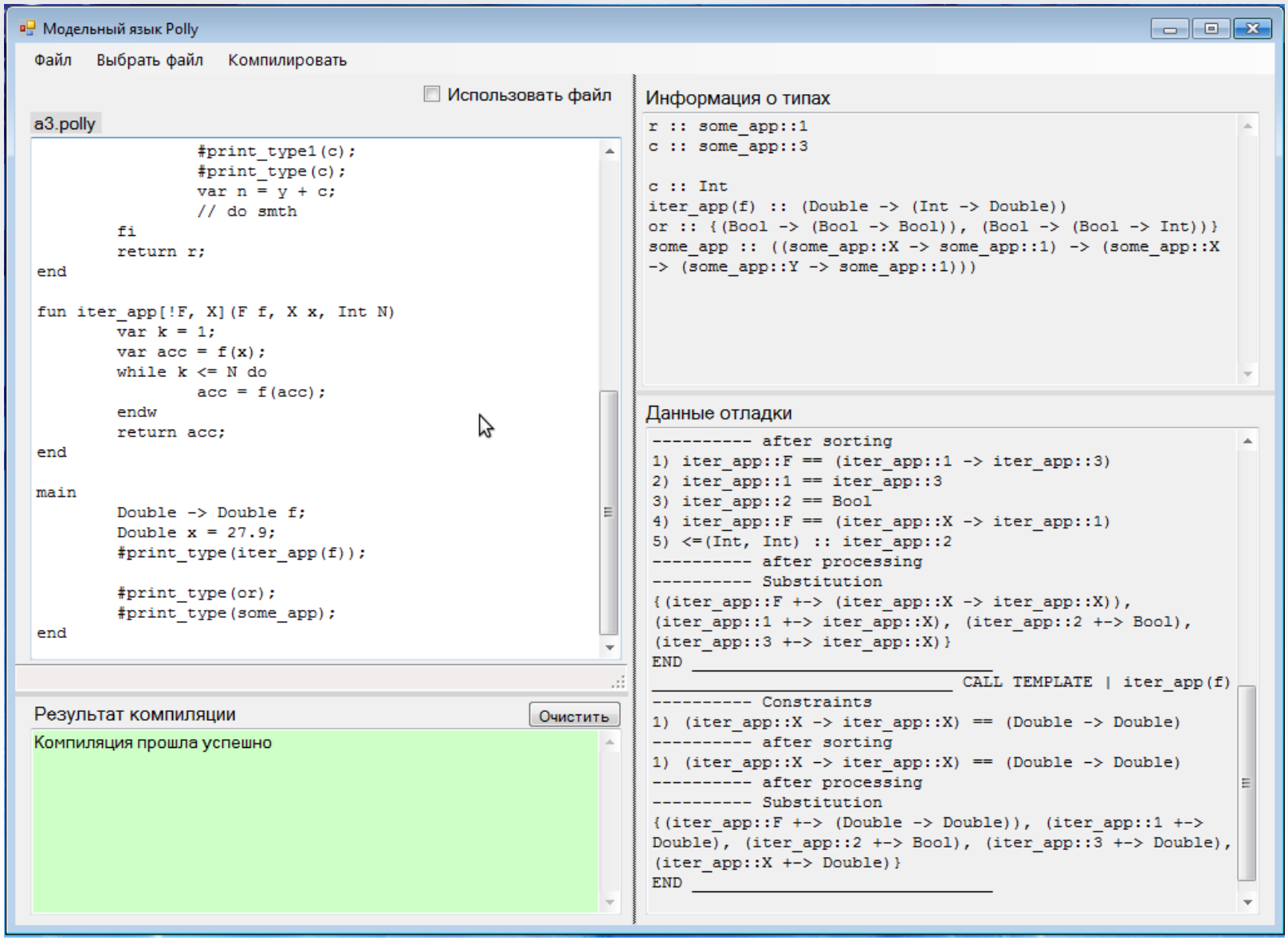
Если существуют корректные конкретизации шаблона, то сохраняется достаточное **множество ограничений**. В противном случае получена ошибка компиляции.

Для **инстанцирования шаблона** нужно:

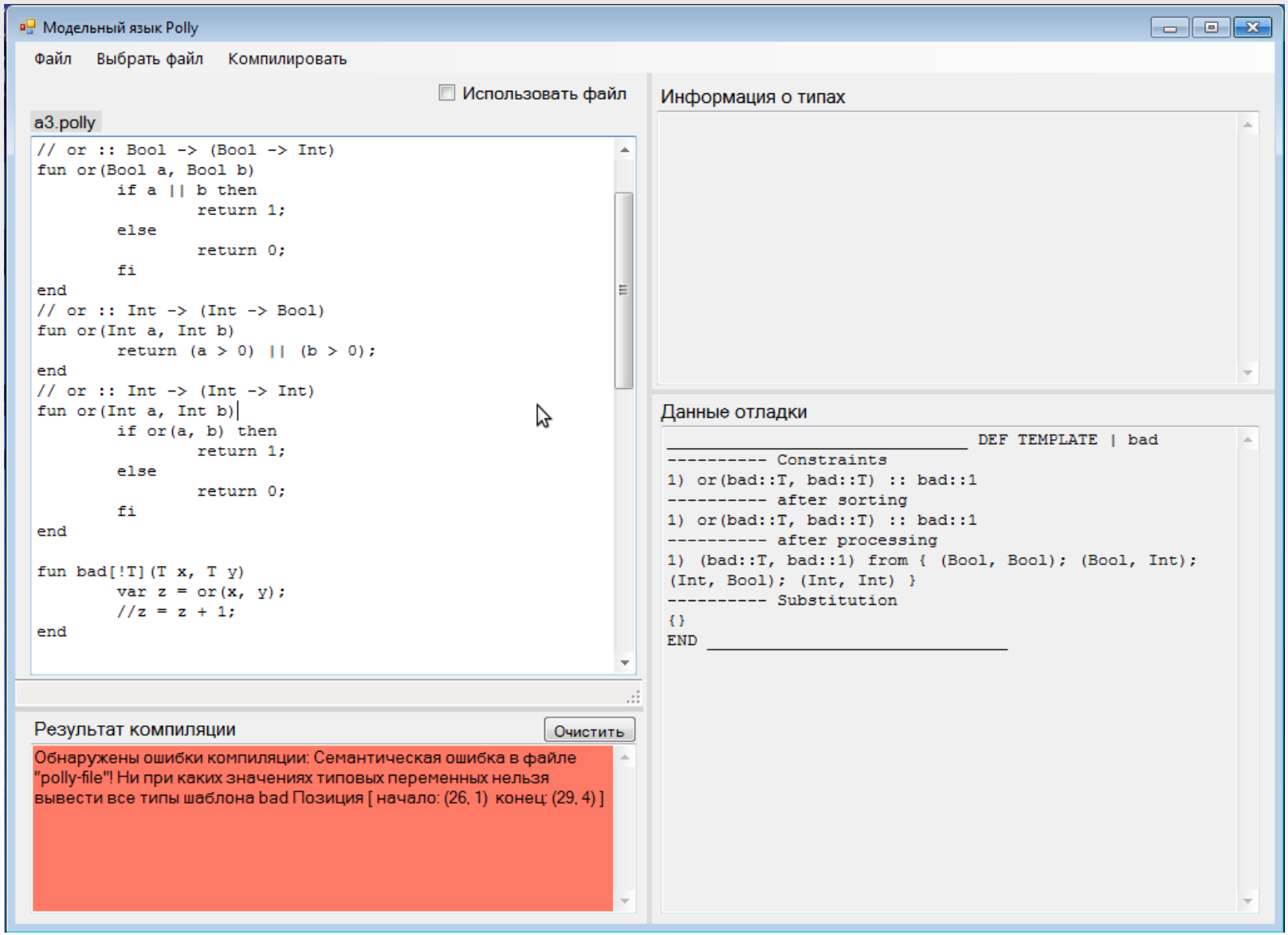
- сопоставить конкретные типы параметрам шаблона;
- проверить набор ограничений.

Содержание

Пример правильной программы



Пример неправильной программы



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.