

# Язык и компилятор Рефал-5λ

Александр Коновалов

МГТУ имени Н. Э. Баумана

Совместное рабочее совещание  
ИПС имени А. К. Айламазяна РАН  
и  
МГТУ имени Н. Э. Баумана

5 июня 2018 г.

# История проекта

- 2008 — Простой Рефал. Компилятор подмножества базисного Рефала в синтаксисе, близком к Рефалу-5 в C++
- 2009 — начало использования Простого Рефала как тестового полигона
- 2009 — появление безымянных вложенных функций в языке
- 2015—2017 — генерация интерпретируемого кода
- 2016 — оптимизации совместного сопоставления с образцом и построения результата
- 2017 — Рефал-5λ

# Простой Рефал как тестовый ПОЛИГОН

- 2008 — Александр Коновалов, компиляция Рефала в императивный код
- 2009 — Вадим Сухарев, интерпретируемый код для результатных выражений (курсовой проект)
- 2009 — Александр Коновалов, вложенные безымянные функции
- 2010 — Наталья Быкадорова (Коннова), компиляция в C# (курсовой проект)
- 2010 — Михаил Дорофеев, попытка оптимизации результатных выражений (курсовой проект)
- 2015 — Игорь Дрогунов, компиляция в интерпретируемый код (курсовой проект)
- 2016 — Евгений Копьёв, оптимизация построения результатных выражений (ВКР бакалавра)
- 2016 — Иван Скрыпников, оптимизация совместного сопоставления с образцом (ВКР бакалавра)
- 2016 — Елена Бурлова, генератор лексических анализаторов (курсовой проект)
- 2016 — Дарья Сухомлинова, пошаговый отладчик (в стиле `reftr`; курсовой проект)
- 2016 — Дамир Габбасов, присваивания (курсовой проект)
- 2017 — Анастасия Козлова, условия (курсовой проект)
- 2018 — Павел Савельев, улучшенная оптимизация совместного сопоставления с образцом (ВКР бакалавра)

# Рефал-5λ: особенности языка и компилятора

- Точное надмножество Рефала-5
- Функции высших порядков
- Абстрактные типы данных — «запечатанные скобки»
- Вкусный и полезный синтаксический сахар
- Классическая списковая реализация
- Исполнимый файл = интерпретатор + RASL
- Оптимизации
- Удобный FFI с языком C++

# Рефал-5λ — точное надмножество Рефала-5

Проектирование и разработка библиотеки встроенных функций Рефала-5λ #102

Edit New issue

 Closed Mazdaywik opened this issue on Jul 27 2017 · 12 comments



Mazdaywik commented on Jul 27 2017 · edited ▾

Member



Эта задача — подзадача для #51.

Эта задача — подзадача для #134.

В рамках данной задачи необходимо сделать следующее.

Во-первых, проанализировать весь список встроенных функций РЕФАЛа-5 (версии PZ Oct 29 2004 ) и для каждой из них описать:


- формат и кратко её семантику,
  - возможный конфликт с одноимённой библиотечной функцией Простого Рефала (что имеет место быть, например, для арифметики),
  - приоритет реализации (первая очередь / подождёт),
  - подходы к реализации и разрешению возможного конфликта.
- Это следует сделать в первом комментарии.

Во-вторых, создать подзадачи на реализации соответствующих функций и/или групп функций.

- ☒ 1. Mu (#135)
- ☒ 2. Add (#130)
- ☒ 3. Arg (#124)
- ☒ 4. Br (#131)
- ☒ 5. Card (#128)
- ☒ 6. Chr (#132)
- ☒ 7. Cp (#131)
- ☒ 8. Dg (#131)

Assignees



 Mazdaywik

Labels



task

Projects




None yet

Milestone



Рефал-5λ

Notifications

 Unsubscribe

You're receiving notifications because you modified the open/close state.

1 participant



 Lock conversation

# Рефал-5λ — точное надмножество Рефала-5

```
✓ 30. Sub (#130)
✓ 31. Symb (#130)
✓ 32. Time (#136)
✓ 33. Type (#127)
✓ 34. Upper (#132)
✓ 35. Sysfun (#138)
✓ 36- Imp$$
✓ 37- Stop$$
✓ 44- (нет-имени)
✓ 45- Freeze
✓ 46- Freezer
✓ 47- Da
✓ 48- Up
✓ 49- Ev-met
✓ 50. Residue (#135)
✓ 51. GetEnv (#124)
✓ 52. System (#136)
✓ 53. Exit (#136)
✓ 54. Close (#128)
✓ 55. ExistFile (#124)
✓ 56. GetCurrentDirectory (#136)
✓ 57. RemoveFile (#136)
✓ 58. Implode_Ext (#132)
✓ 59. Explode_Ext (#102)
✓ 60. TimeElapsed (#136)
✓ 61. Compare (#130)
✓ 62. DeSysfun (#138)
✓ 63- XMLParse
✓ 64. Random (#139)
✓ 65. RandomDigit (#139)
✓ 66. Write (#128)
✓ 67. ListOfBuiltin (#136)
✓ 68. SizeOf (#136)
✓ 69. GetPID (#136)
✓ 70- int4fab_1
✓ 71. GetPPID (#136)
```

# Функции высших порядков в Рефале-5λ

```
*$FROM LibraryEx  
$EXTERN Map;
```

```
PrintEachLine {  
  (e.Line) = <Prout e.Line>;  
}
```

```
$ENTRY PrintLines-1 {  
  e.Lines = <Map &PrintEachLine e.Lines>;  
}
```

```
$ENTRY PrintLines-2 {  
  e.Lines = <Map { (e.Line) = <Prout e.Line>; } e.Lines>;  
}
```

# Функции высших порядков в Рефале-5λ

```
Map-1 {  
  s.Func t.Item e.Items  
    = <Mu s.Func t.Item> <Map-1 s.Func e.Items>;  
  
  s.Func /* пусто */ = /* пусто */;  
}
```

```
Map-2 {  
  s.Func t.Item e.Items  
    = <s.Func t.Item> <Map-1 s.Func e.Items>;  
  
  s.Func /* пусто */ = /* пусто */;  
}
```



# Абстрактные типы данных

```
$ENUM SymTable; /* эквивалентно SymTable {} */

* <SymTable-Create> == t.SymTable
$ENTRY SymTable-Create {
    = [SymTable];
}

* <SymTable-Lookup t.SymTable e.Name>
*   == Success e.Value
*   == Fails
$ENTRY SymTable-Lookup {
    [SymTable e.Names-B ((e.Name) e.Value) e.Names-E] e.Name = Success e.Value;
    [SymTable e.Names] e.Name = Fails;
}

* <SymTable-Update t.SymTable (e.Name) e.Value>
*   == t.SymTable
$ENTRY SymTable-Update {
    [SymTable e.Names-B ((e.Name) e.OldValue) e.Names-E] (e.Name) e.NewValue
        = [SymTable e.Names-B ((e.Name) e.NewValue) e.Names-E];

    [SymTable e.Names] (e.Name) e.Value
        = [SymTable e.Names ((e.Name) e.Value)];
}
```

# Синтаксический сахар: блоки

Блок можно приписать к любому результатному выражению (в том числе, в условии)

```
Foo {  
    некоторый образец  
    , условие  
    : { ...блок 1... }  
    : { ...блок 2... }  
    : образец условия  
    = результатное выражение;  
}
```

# Синтаксический сахар: блоки

Блок выражается через вложенные функции:

`Result : { ...A... } : { ...B... } : { ...C... }`

является эквивалентом для

`<{ ...C... } <{ ...B... } <{ ...A... } Result>>>`

# Синтаксический сахар: присваивания

Присваивание — «безоткатное условие»:

```
Length {  
  e.Expr  
    = <Lenw e.Expr> : s.Len e.Expr1  
    = s.Len  
}
```

Присваивание выражается через блок:

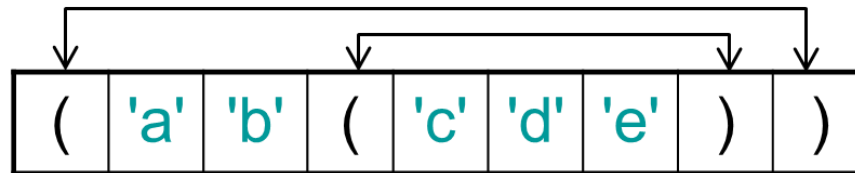
```
Length {  
  e.Expr  
    = <Lenw e.Expr>  
    : {  
      s.Len e.Expr1 = s.Len  
    }  
}
```

# Синтаксический сахар: сокрытие переменных

```
/* Procedure → Header Declarations Body. */  
ParseProcedure {  
    e.Tokens  
    = <ParseHeader e.Tokens>  
      : (e.ProcedureName) (e.Parameters) (e.HeaderErrors) e.Tokens^  
  
    = <ParseDeclarations e.Tokens>  
      : (e.Declarations) (e.DeclErrors) e.Tokens^  
  
    = <ParseBody e.Tokens> : (e.Body) (e.BodyErrors) e.Tokens^  
  
    = ((e.ProcedureName) (e.Parameters) (e.Declarations) e.Body)  
      (e.HeaderErrors e.DeclErrors e.BodyErrors)  
      e.Tokens;  
}
```

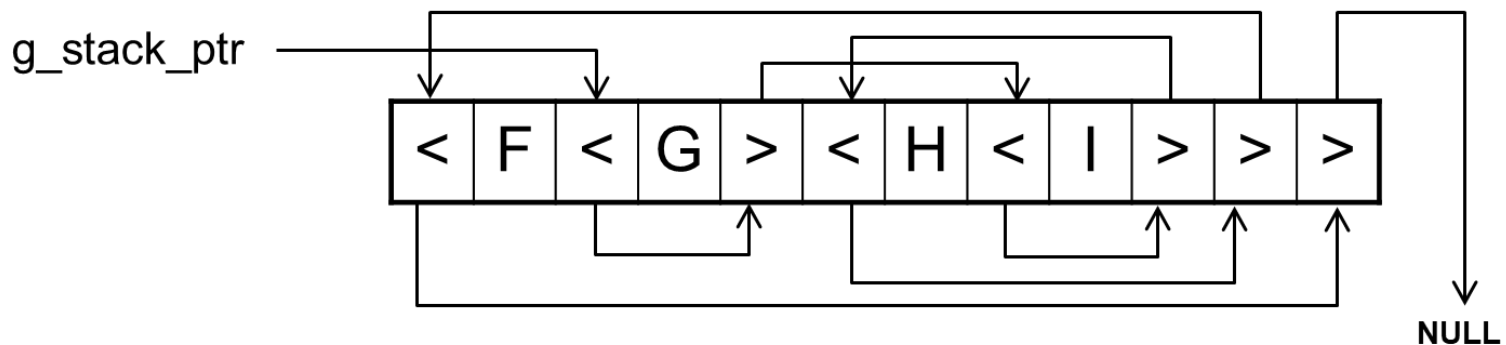
# Классическая списковая реализация

## Представление структурных скобок

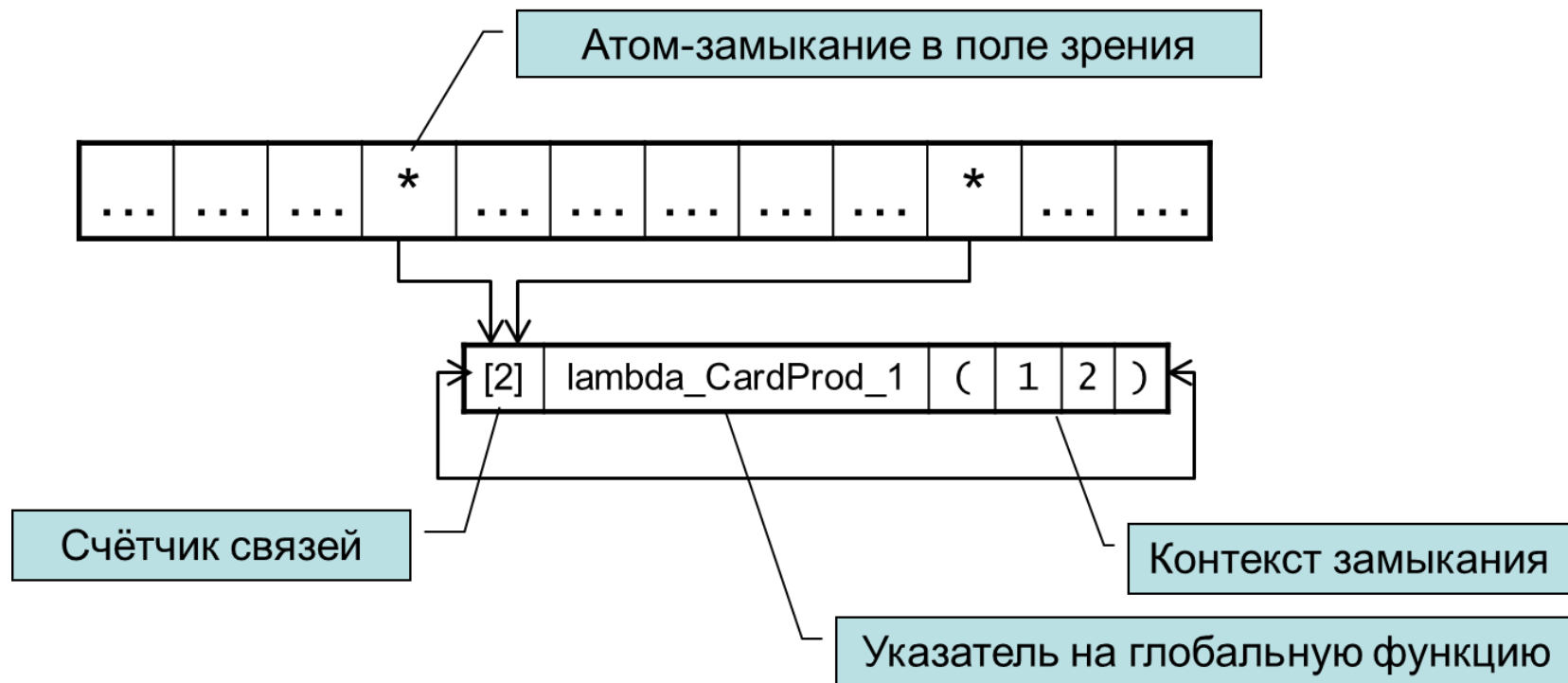


## Представление угловых скобок

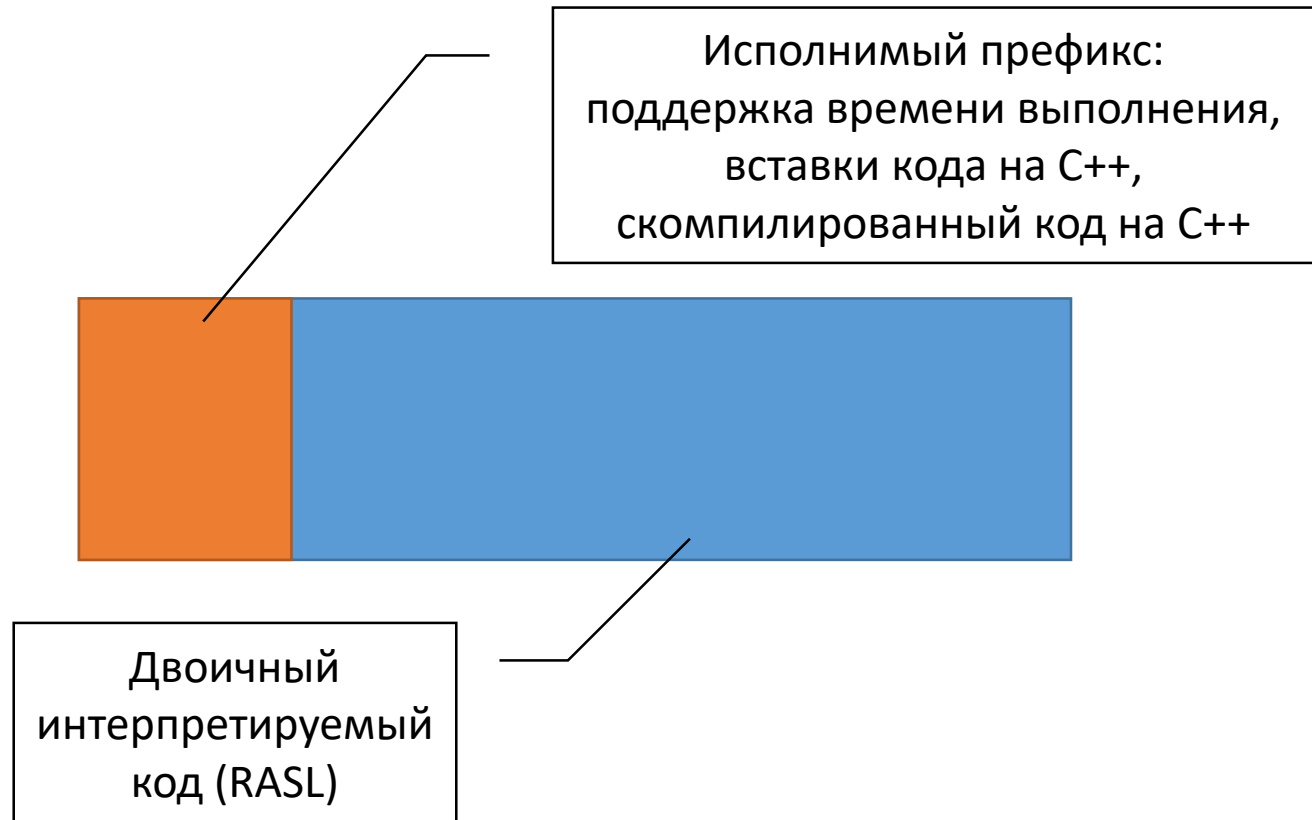
Угловые скобки образуют односвязный список, на голову которого указывает глобальная переменная `g_stack_ptr`.



# Классическая списковая реализация



# Структура исполнимых файлов





# Оптимизация совместного сопоставления с образцом

## Изменения в сгенерированном коде

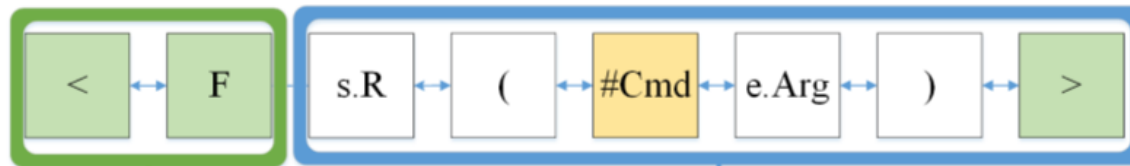
До оптимизации	После оптимизации
<pre>do {     if( !char_left( 'a' ))         continue;      if( !char_left( 'b' ))         continue;  } while ( 0 );  do {     if( !char_left( 'a' ))         continue;      if( !char_left( 'd' ))         continue;  } while ( 0 );</pre>	<pre>if( !char_left( 'a' ))     return cRecognitionImpossible;  do {     if( !char_left( 'b' ))         continue;  } while ( 0 );  do {     if( !char_left( 'd' ))         continue;  } while ( 0 );</pre>

Слайд из презентации Ивана Скрыпникова

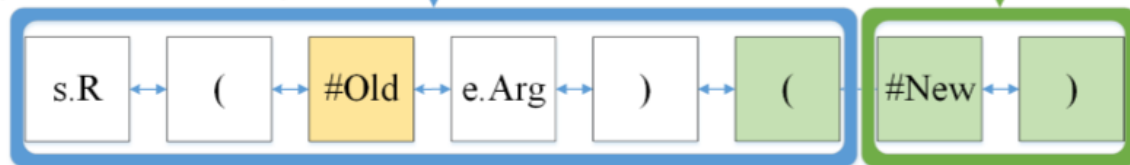
# Оптимизация построения результатного выражения



## Пример найденного перекрытия

Вызов функции  
(исходный список):



Результат вызова  
(конечный список):



-  - значение было изменено.
-  - тэг и значение были изменены.

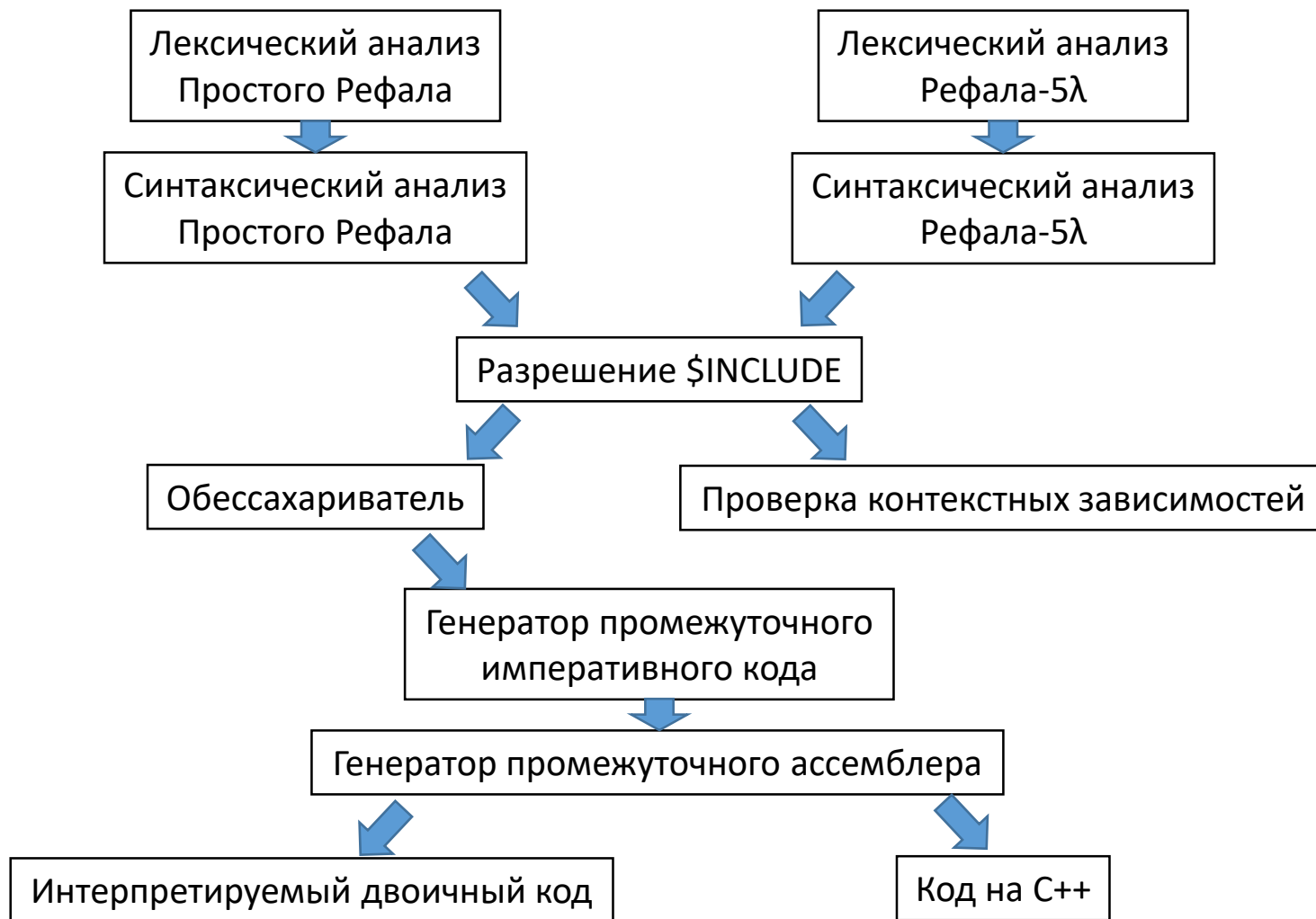
Слайд из презентации Евгения Копьёва

# Интерфейс с языком C++:

## НАТИВНЫЕ ВСТАВКИ

```
%%  
// это нативная вставка в глобальной области видимости  
#include <stdio.h>  
  
static int g_next_number = 0;  
%%  
  
$ENTRY NextNumber {  
%%  
    // это нативная вставка внутри тела функции, т.е. функция целиком пишется на C++.  
  
    refalrts::Iter content_b = 0, content_e = 0;  
    refalrts::Iter pfunc_name = refalrts::call_left(content_b, content_e, arg_begin, arg_end);  
  
    if (! refalrts::empty_seq(content_b, content_e)) {  
        return refalrts::cRecognitionImpossible;  
    }  
  
    ++g_next_number;  
    printf("Generating next number %d\n", g_next_number);  
  
    refalrts::reinit_number(arg_begin, g_next_number);  
    refalrts::splice_to_free_list(pfunc_name, arg_end);  
    return refalrts::cSuccess;  
%%  
}
```

# Архитектура компилятора



Спасибо  
за внимание!

# Ранний Простой Рефал

```
// Объявления библиотечных функций
$EXTERN ReadLine, WriteLine;
// Объявление локальной функции
$FORWARD Fab;
// Точка входа в программу
$ENTRY Go {
    =
        <WriteLine
            <Fab <ReadLine>>
        >;
}

Fab {
    e.Begin 'a' e.End = e.Begin 'b' <Fab e.End>;

    e.Other = e.Other;
}
```

# Ранний Простой Рефал

```
// Функция Map преобразует каждый терм выражения согласно заданному правилу  
// Вызов <Map s.Trans e.Elems> == e.Transformed
```

```
$ENTRY Map {  
  s.Trans t.First e.Tail = <s.Trans t.First> <Map s.Trans e.Tail>;  
  
  s.Trans = /* пусто */;  
}
```

```
// Функция CardProd вычисляет декартово произведение двух множеств  
// Вызов <CardProd ('a' 'b' 'c') (1 2)>  
// == ('a' 1) ('a' 2) ('b' 1) ('b' 2) ('c' 1) ('c' 2)
```

```
$ENTRY CardProd {  
  (e.SetA) (e.SetB) =  
    <Map  
      {  
        t.A =  
          <Map  
            { t.B = (t.A t.B); }  
            e.SetB  
          >;  
        }  
      e.SetA  
    >;  
}
```