



Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Московский государственный технический университет  
имени Н.Э. Баумана  
(национальный исследовательский университет)»  
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ \_\_\_\_\_

КАФЕДРА \_\_\_\_\_

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**  
***К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ***  
***НА ТЕМУ:***

***«Актуализация SCP3»***

Студент \_\_\_\_\_  
(Группа)

\_\_\_\_\_  
(Подпись, дата) (И.О.Фамилия)

Руководитель ВКР

\_\_\_\_\_  
(Подпись, дата) (И.О.Фамилия)

Консультант

\_\_\_\_\_  
(Подпись, дата) (И.О.Фамилия)

Консультант

\_\_\_\_\_  
(Подпись, дата) (И.О.Фамилия)

Нормоконтролер

\_\_\_\_\_  
(Подпись, дата) (И.О.Фамилия)

2019 г.



## АННОТАЦИЯ

Тема: «Актуализация SCP3».

Цель данной работы – провести актуализацию частично самоприменимого суперкомпилятора SCP3.

Для достижения поставленной цели было произведено ознакомление с языком РЕФАЛ-5 и изучены принципы работы суперкомпилятора SCP3. Была разработана утилита, реализующая преобразование программ на РЕФАЛе-5 в соответствии с изменениями, внесенными в данный язык программирования. Также с помощью данной утилиты перенесена под актуальную версию языка РЕФАЛ-5 реализация суперкомпилятора SCP3.

Объем выпускной квалификационной работы составляет 42 страницы, на которых размещены 3 рисунка, 1 таблица и 24 листинга. Для написания работы использовались 11 источников.

# СОДЕРЖАНИЕ

<b>СОДЕРЖАНИЕ</b> .....	<b>4</b>
<b>ВВЕДЕНИЕ</b> .....	<b>6</b>
<b>2 ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ</b> .....	<b>8</b>
2.1 Язык РЕФАЛ-5 .....	8
2.1.1 Лексические единицы языка РЕФАЛ-5 .....	8
2.1.2 Синтаксис языка РЕФАЛ-5 .....	10
2.1.3 Семантика языка РЕФАЛ-5 .....	10
2.1.4 Изменения, внесенные в РЕФАЛ-5 .....	12
2.2 Суперкомпиляция .....	12
2.3 Суперкомпилятор SCP3 .....	13
2.3.1 Плоский РЕФАЛ .....	14
2.3.2 Граф сопоставления с образцом .....	16
2.3.3 Метасистемный переход .....	17
<b>3 РАЗРАБОТКА</b> .....	<b>21</b>
3.1. Разработка лексического анализатора.....	21
3.2. Разработка синтаксического анализатора.....	22
3.3. Разработка семантического анализатора .....	22
3.4. Разработка преобразователя промежуточного представления программы в код на языке РЕФАЛ-5.....	23
<b>4 РЕАЛИЗАЦИЯ</b> .....	<b>25</b>
4.1 Реализация лексического анализатора .....	25
4.2 Реализация синтаксического анализатора .....	27
4.3. Реализация семантического анализатора.....	29
4.4. Реализация преобразователя промежуточного представления программы в код на языке РЕФАЛ-5.....	30
<b>5 ТЕСТИРОВАНИЕ</b> .....	<b>31</b>
5.1. Суперкомпиляция ленивого интерпретатора.....	31
5.2 Суперкомпиляция интерпретатора .....	34
<b>6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ</b> .....	<b>36</b>
6.1. Установка утилиты и запуск .....	36

6.2	Использование утилиты .....	37
6.3	Использование суперкомпилятора SCP3 .....	39
<b>ЗАКЛЮЧЕНИЕ .....</b>		<b>40</b>
<b>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....</b>		<b>41</b>
<b>ПРИЛОЖЕНИЕ А .....</b>		<b>42</b>

## ВВЕДЕНИЕ

Одним из методов оптимизации алгоритмов является суперкомпиляция - специальная техника автоматической специализации программ. В 1996 году Андреем Немытых, Викторией Пинчук и Валентином Турчиным был разработан модельный суперкомпилятор SCP3 для языка программирования РЕФАЛ-5, а также проведены простейшие эксперименты по самоприменению.

В начале двухтысячных годов синтаксис языка РЕФАЛ-5 был существенно изменён без сохранения обратной совместимости, из-за чего дальнейшая разработка и модификации суперкомпилятора SCP3 стали затруднительны.

Чтобы появилась возможность дальнейшего развития данного суперкомпилятора, необходимо было провести его актуализацию.

## 1 ПОСТАНОВКА ЗАДАЧИ

Целью данной квалификационной работы является актуализация частично самоприменимого суперкомпилятора SCP3. Под актуализацией суперкомпилятора в данном конкретном случае подразумевается перенос его реализации под актуальную версию языка программирования РЕФАЛ-5. Для достижения данной цели были поставлены следующие задачи:

1. Ознакомление с языком РЕФАЛ-5 и изучение принципов работы суперкомпилятора;
2. Реализация утилиты, осуществляющей трансформацию программы в соответствии с изменениями, внесенными в синтаксис языка РЕФАЛ-5;
3. Преобразование исходного кода суперкомпилятора SCP3 с помощью вышеописанной утилиты;
4. Внесение изменений в лексический и синтаксический анализаторы суперкомпилятора SCP3 с целью его адаптации под новый синтаксис языка РЕФАЛ-5;
5. Проведение тестов с целью проверки работоспособности суперкомпилятора SCP3;

## 2 ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

### 2.1 Язык РЕФАЛ-5

Язык РЕФАЛ относится к классу функциональных языков программирования. Особенностью этого языка является ориентация на символьные преобразования, в числе которых обработка символьных строк, анализ и преобразование исходных текстов программ и т.д. РЕФАЛ Был разработан в 1966 году В.Ф. Турчиным [1]. На сегодняшний день существует несколько диалектов языка (РЕФАЛ-2 (1970-е), РЕФАЛ-5 (1985), РЕФАЛ+ и т.д.). Фактическим стандартом среди них признан РЕФАЛ-5, на котором был реализован суперкомпилятор SCP3. Для анализа внесенных изменений в РЕФАЛ-5 стоит остановиться на рассмотрении устаревшего стандарта этого диалекта подробнее.

#### 2.1.1 Лексические единицы языка РЕФАЛ-5

Лексические единицы языка РЕФАЛ-5 разделяют на специальные знаки, системные ключевые слова, символы, переменные, комментарии, а также разделители.

Специальными знаками являются:

1. Одинарная и двойные кавычки;
2. Круглые скобки;
3. Вычислительные скобки;
4. Фигурные скобки
5. Индикаторы типа переменной 's', 't', 'e';
6. Знак равенства;
7. Точка с запятой;
8. Двоеточие;
9. Запятая;
- 10.Точка;

К системным ключевым словам относятся \$ENTRY и \$EXTERNAL. Для последнего предусмотрены в качестве альтернативы слова \$EXTERN или \$EXTRN.

Символами в языке РЕФАЛ-5 называют минимальные синтаксические элементы структур данных. В качестве символов выделяют:

1. Символические имена, называемые идентификаторами. Они могут содержать латинские буквы, цифры и знаки “-” и “\_”, но должны обязательно начинаться с заглавной буквы;
2. Целые числа;
3. Литеры, заключаемые в кавычки;

Для представления кавычек как литер необходимо их экранировать. Таким образом, две одинарные или двойные кавычки подряд обозначают литеру-кавычку одинарную или двойную соответственно.

Переменная представляет собой индикатор типа, за которым следуют точка и индекс. Индекс, в свою очередь, может быть идентификатором либо числом. В случае, когда индекс представлен односимвольным идентификатором или цифрой, точка может быть опущена. Если после индикатора типа присутствует точка, то идентификатор может начинаться со строчной буквы.

Тип переменной определяет ее свойства:

1. ‘s’ – символьная переменная;
2. ‘t’ – переменный терм;
3. ‘e’ – переменное выражение.

Терм представляется в виде символа или выражения, заключенного в структурные скобки.

Комментарии делятся на:

1. Строковые комментарии – строки, начинающиеся с символа “\*”;
2. Комментарии-вставки, представляющие собой произвольную последовательность, заключенную в пару “/\*” и “\*/”.

В группу разделителей принято выделять знаки пробела, табулирования и возврата каретки.

Стоит упомянуть, что в первоначальной версии языка РЕФАЛ-5 имена функций, индексы переменных и символьные имена были нечувствительны к регистру. Соответственно, переменные e.x и e.X считались эквивалентными.

### 2.1.2 Синтаксис языка РЕФАЛ-5

Программа на языке РЕФАЛ-5 представляется в виде перечня функциональных определений и объявления внешних функций. Грамматика РЕФАЛа-5 представлен в приложении А.

Объявление внешних функций начинается с ключевого слова `$EXTERNAL` (`$EXTERN` `$EXTRN`), за которым следуют имена функций, разделенные запятой. Для отделения одного внешнего объявления от другого используется знак `‘;’`. Этот же знак может разделять функциональные определения.

Определения функции, в свою очередь, начинаются с названия функции, за которым следует конструкция, представляющая собой упорядоченный набор предложений, заключенный в фигурные скобки. Подобная конструкция называется блоком. Перед именем функции может присутствовать ключевое слово `$ENTRY`.

Предложение в языке РЕФАЛ-5 принято разделять на левую и правую части. Левая часть содержит выражение-образец, а правая может включать различную комбинацию условных конструкций или общее выражение.

Условная конструкция начинается со знака запятой, за которым следует сопоставляемая пара, где в качестве левого операнда (аргумента) может выступать произвольное РЕФАЛ-выражение, а в качестве правого (образца) – произвольное выражение-образец или блок.

Объектным выражением называется выражение, содержащее произвольный набор символов, которые, в свою очередь, могут быть заключены в структурные скобки.

Выражение, включающее символы, структурные скобки и свободные переменные, но не содержащее вызовов функций называют выражением-образцом или образцом.

### 2.1.3 Семантика языка РЕФАЛ-5

Семантика РЕФАЛа-5 основана на сопоставлении с образцом.

Сопоставление объектного выражения *Е* (аргумента сопоставления), образцу *Р* обозначается как *Е:Р*. Результатом успешного сопоставления является подстановка значений вместо переменных в *Р*, в следствие чего *Р* становится тождественным *Е*.

Как было упомянуто выше, функции содержат перечень предложений. Выполнение функции состоит в поочередном сопоставлении аргумента функции выражению-образцу каждого из предложений. Как было указано выше, в предложении может быть несколько условных конструкций, следующих друг за другом. В таком случае условия будут выполняться в заданной последовательности. Если очередное сопоставление проходит успешно, то новое выражение, сформированное на основании образцов соответствующего предложения и содержащихся в нем условных конструкций, будет являться результатом функции. В случае, если на этапе поочередного сопоставления не был найден такой образец, с которым сопоставления произошло бы успешно, то фиксируется ошибка и программа аварийно завершается.

Пример программы на устаревшем стандарте языка РЕФАЛ-5 приведен на Листинге 1. В данном примере конструкцией “\$ENTRY Go” обозначается точка входа в программу, а символ-метка “Prout” обозначает функцию, производящую запись своего аргумента в стандартный поток вывода. Функция Palindrom производит проверку, является ли ее аргумент палиндромом. При запуске данной программы в стандартном потоке вывода появится строка “True”.

Листинг 1. Пример программы на устаревшем стандарте языка РЕФАЛ-5

```
$ENTRY Go {  
    = <Prout <Palindrom 'abcba'>>;  
}  
Palindrom {  
    s1 e1 s1 = <Palindrom e1>;  
    s1 = True;  
    /* пусто */ = True;  
    e1 = False;}
```

#### 2.1.4 Изменения, внесенные в РЕФАЛ-5

В начале двухтысячных годов язык РЕФАЛ-5 был изменен без поддержки обратной совместимости. В перечень изменений, из-за которых была нарушена обратная совместимость, попали следующие изменения:

1. Семантика одинарных и двойных кавычек была изменена:
  - a. набор символов, заключенных в двойные кавычки, стал считаться единым символом;
  - b. символы одинарной и двойной кавычек, обратной косой черты, перевода строки, табуляции и возврата каретки необходимо экранировать символом обратной косой черты;
2. Были запрещены имена переменных вида  $e1\ tX\ s2$  – теперь наличие точки между индикатором типа и индексом обязательно;
3. Идентификаторы и символы стали чувствительны к регистру, кроме того, были разрешены идентификаторы, начинающиеся с букв в нижнем регистре;
4. Была изменена семантика встроенной функции Type.

#### 2.2 Суперкомпиляция

Целью суперкомпиляции является построение конечного графа конфигураций. Конфигурациями называют множества состояний вычислительного процесса, и они располагаются в вершинах описанного графа [8]. В качестве дуг выступают всевозможные переходы между множествами состояний.

В основе суперкомпиляции лежат следующие операции [5] [7]:

1. Прогонка;
2. Обобщение;
3. Зацикливание;

Прогонка – это процесс порождения новых конфигураций из уже имеющихся. Применение лишь этой операции может привести к построению бесконечного дерева. Так как в результате суперкомпиляции должен быть

получен конечный граф конфигураций, необходимо применение двух других операций: обобщения и зацикливания.

Обобщением называют замену одной конфигурации на другую, представляющую больше состояний, чем исходная конфигурация.

Зацикливание заключается в сведении конфигурации к эквивалентной (уже встречавшейся ранее).

Полученный в итоге конечный граф конфигураций преобразуется в остаточную программу. Особенностью остаточной программы является то, что в ней могут присутствовать не все части исходной программы. Некоторые из них могут быть удалены в результате суперкомпиляции, например, если они недостижимы при заданных ограничениях на входные данные.

### 2.3 Суперкомпилятор SCP3

Как упоминалось ранее, частично самоприменимый суперкомпилятор SCP3 был реализован в 1996 году. Для реализации самоприменимости при разработке данного суперкомпилятора пришлось ввести следующий набор ограничений [4]:

1. В качестве входного языка было выбрано подмножество РЕФАЛа, именуемое Плоским РЕФАЛом;
2. Процесс прогонки [11], который играет ключевую роль построении трансформированной программы, был реализован в виде преобразования плоского графа сопоставления с образцом;
3. Были включены метасистемные прыжки, как описано в [2]. Они позволяют переложить работу по просчёту транзитных шагов с суперкомпилятора на интерпретатор, выполняющий РЕФАЛ.

Для пользователя суперкомпилятор представляется в виде черного ящика, на вход которому в качестве аргументов подаются задание на суперкомпиляцию в виде MST-схемы и набор функций и выражений, над которыми необходимо провести суперкомпиляцию. Результатом работы суперкомпилятора является остаточная программа. Описанная схема приведена на Рисунке 1.

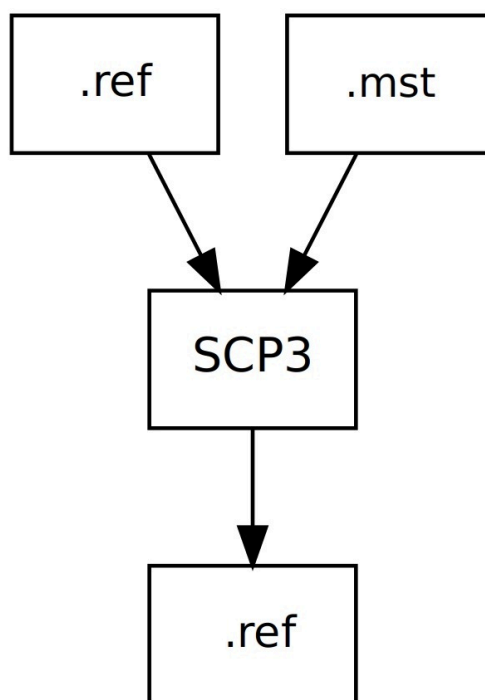


Рисунок 1. Упрощенная схема суперкомпиляции

### 2.3.1 Плоский РЕФАЛ

Для достижения самоприменимости суперкомпилятора было необходимо максимально упростить входной язык программирования. В связи с этим соображением был разработан Плоский РЕФАЛ. Он имеет две формы записи:

1. Сентенциальная форма - запись в виде предложений, удобная для чтения человеком;
2. Форма графа сопоставления с образцом, подходящая для автоматической обработки.

Плоский РЕФАЛ находится на самом нижнем уровне в иерархии существующих диалектов языка РЕФАЛ. Это означает, что синтаксис этого языка является самым простым и при этом ограниченным.

Синтаксис Плоского РЕФАЛа приведен на листинге 2.

## Листинг 2. Синтаксис плоского РЕФАЛа.

```

symbol      ::= symbolic-name | number | 'character'
expression  ::= [] | term expression
[]           ::=
term         ::= symbol | variable | ( expression )
rigid-pattern ::= [] | e.index | rigid-term rigid-pattern
              | rigid-pattern rigid-term
rigid-term  ::= s.index | symbol | ( rigid-pattern )
variable    ::= s.index | e.index
index       ::= number | symbolic-name
function-def ::= fn-name { sentences }
fn-name     ::= symbolic-name
sentences   ::= sentence | sentence sentences
sentence    ::= left-side = right-side ;
left-side   ::= rigid-pattern
right-side  ::= expression | function-call
function-call ::= < fn-name expression >
program     ::= sentence | sentence program

```

Основным отличием Плоского РЕФАЛа от РЕФАЛ-5 являются особые правила записи предложения. В предложениях Плоского РЕФАЛа запрещены условные конструкции и блоки, введенные в стандарт Расширенного РЕФАЛа. Левая часть предложения в плоском подмножестве должна быть представлена жестким образцом, то есть таким выражением, что ни одно из его подвыражений не принимает форму  $E_1 e.i_1 E_2 e.i_2 E_3$ , где  $E_1, E_2, E_3$  – произвольные выражения. Кроме того, в образце не должно быть повторных вхождений ‘e’-переменных. Правая же часть предложения в плоской версии языка РЕФАЛ должна быть представлена либо в виде объектного выражения, либо в виде единичного вызова функции. Вложенные вызовы функций и комбинации (объектных?) выражений с вызовами функций в правых частях предложений запрещены.

В Плоском РЕФАЛе обмен информацией происходит только через переменные, а не через значения, принимаемые вызовами функций. Это итеративный, а не рекурсивный стиль программирования.

### 2.3.2 Граф сопоставления с образцом

Программа, написанная на Плоском РЕФАЛе автоматически конвертируется суперкомпилятором SCP3 в плоский граф сопоставления с образцом [9].

В введенных обозначениях сопоставления с образцом  $E_0 : P$  левая часть  $E_0$  должна быть объектным выражением. Теперь обобщим эту концепцию, допустив наличие в левой части произвольного выражения  $E$ . В таком случае переменные в  $E$  должны быть связанными, то есть иметь определенные значения. Выполнение такого обобщенного сопоставления происходит за два шага:

1. Подстановка вместо связанных переменных в  $E$  их значений, результатом чего становится некоторое объектное выражение  $E_0$ ;
2. Выполнение сопоставления  $E_0 : P$ , где левая часть теперь является объектным выражением.

В результате успешного сопоставления свободные переменные в выражении-образце  $P$  получают определенные значения. Если после этого в образце не осталось переменных, то есть сопоставление приняло вид  $a:a$ , результат будет обозначен как  $I$  (тождественная операция). Когда сопоставление завершается неудачей, результат обозначается как  $Z$  (невозможная операция).

Граф сопоставления с образцом представляет собой в общем виде дерево, содержащее возможные пути вычислений. Его принято записывать в синтаксисе, аналогичном синтаксису арифметических выражений:

1. Термы в сумме обозначают дуги графов, имеющие общий начальный узел;
2. Конкатенация термов представляет последовательное выполнение базовых операций;
3. Дополнение или ветвление определяет наличие различные возможные случаев.

Решения о том, какой из путей должен быть выбран, регулируются сужениями (сопоставление с образцом  $v:P$ , где  $v$  – переменная, а  $P$  – жесткий образец) и рестрикциями, последние из которых являются, по существу, отрицательными сужениями. Благодаря использованию рестрикций различные дуги, исходящие из одной вершины графа, могут обрабатываться независимо друг от друга.

### 2.3.3 Метасистемный переход

Введем трехуровневую иерархию  $f_0, f_1, f_2$ . В рамках данной иерархии функция  $f_n$  на уровне  $n$  преобразуется функцией  $f_{n+1}$ , которая находится на уровне  $n + 1$  и, таким образом, представляет собой метасистему по отношению к  $f_n$ . Создание каждого такого нового уровня называется метасистемным переходом [10].

Областью определения функции, описанной на языке РЕФАЛ-5, является множество объектных выражений. Программы на РЕФАЛе, однако, могут включать более общие выражения, включая вычислительные скобки и свободные переменные, следовательно, реализация программы, напрямую манипулирующей другими программами на РЕФАЛе, невозможна. Для обхода этого ограничения вводится отображение множества общих РЕФАЛ-выражений на множество объектных выражений и использование образов «горячих» объектов, то есть свободных переменных и вычислительных скобок, вместо самих объектов.

Подобное отображение называется метакодированием или погружением в метакод. Метакодирование  $E$  обозначается как  $\mu\{E\}$ . Если  $E$  синтаксически представляет собой единичный терм, фигурные скобки могут быть опущены. Правила метакодирования различных выражений, сформулированных в [3], приведены в Таблице 1, где  $S$  обозначает любой символ, а  $i$  – индекс переменной.

Таблица 1. Метакодирование

$E$	$\mu\{E\}$
$[ \ ]$	$[ \ ]$
$S$	$S$
$s.i$	$(s'i)$
$e.i$	$(e'i)$
$(E)$	$(\ast \mu\{E\})$
$< E >$	$(! \mu\{E\})$
$E_1 E_2$	$\mu\{E_1\} \mu\{E_2\}$

В введенной выше иерархии  $f_1$  содержит метакодированный вызов функции  $f_0$  в качестве аргумента, а  $f_2$  – погруженный в метакод вызов функции  $f_1$ . Для более ясного представления подобных иерархий используются многоуровневые MST-схемы. MST-схемы строятся согласно следующему правилу: если подвыражение принимает вид  $E_1 \mu\{E_2\} E_3$ , метакодированная часть перемещается на один уровень ниже и замещается многоточием на основном уровне. Иллюстрация описанной схемы приведена на Рисунке 2.

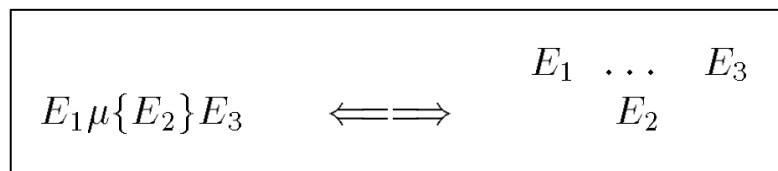


Рисунок 2. Двухуровневая MST-схема

Части РЕФАЛ-выражения, принадлежащие разным метасистемным уровням располагаются на разных строчках. Кроме того, можно поднимать некоторые переменные на один или несколько метасистемных уровней, оставляя на предыдущем месте знак восклицательного знака. Подобные действия радикально изменяют способ выполнения выражения, представленного MST-схемой.

Для правильной интерпретации MST-схем используется следующее правило двух уровней: переменные на верхнем уровне являются свободными.

Они изображают аргументы функции, представленной MST-схемой в целом и должны быть заменены конкретными значениями до проведения вычислений. Переменные, находящиеся на уровень ниже, представляют собой аргументы функции, определение которой является результатом вычислений на верхнем уровне.

В качестве примера рассмотрим широко известную проблему компиляции. Пусть  $\langle L, \text{program}, \text{data} \rangle$  - интерпретатор языка  $L$ , реализованный на РЕФАЛе. Пусть  $P$  – программа на языке  $L$ . Используя суперкомпилятор, мы можем перевести  $P$  в эффективную программу на РЕФАЛе, используя частичные вычисления, согласно MST-схеме, представленной на Листинге 3.

Листинг 3. MST-схема суперкомпиляции интерпретатора  $L$

$\langle \text{Scp} \dots\dots\dots \rangle$ $\langle L \ P \ e.\text{data} \rangle$
---

Предположим, теперь нужна функция, которая на вход принимала бы любую произвольную программу, не только  $P$ . Просто заменив  $P$  на переменную  $e.\text{Program}$  этого не получится достичь. В данном случае программа и переменная находятся на одном уровне и обрабатываются одним и тем же образом. Воспользовавшись правилом двух уровней, можно понять, что результатом в данном случае будет функция от двух переменных: тот же самый интерпретатор. Никаких частичных вычислений производиться не будет, потому что значение переменной  $e.\text{Program}$  остается неизвестным. Чтобы добиться выполнения частичных вычислений, необходимо поднять переменную  $e.\text{Program}$  на один уровень вверх. Данная схема приведена на Листинге 4.

Листинг 4. Поднятие переменной

$\langle \text{Scp} \dots\dots e.\text{Program} \dots\dots \rangle$ $\langle L \dots\dots ! \dots\dots e.\text{data} \rangle$
--

Теперь из правила двух уровней следует, что данной функцией будет ожидаться произвольная программа в качестве значения переменной  $e.\text{Program}$ .

Затем будут проведены частичные вычисления, возвращая программу для функции переменной `e.data`. Функция, определенная данной MST-схемой может в дальнейшем сама стать объектом трансформации для другого частичного вычислителя, результатом чего станет компилятор языка L.

### 3 РАЗРАБОТКА

Одной из поставленных в ходе данной выпускной квалификационной работы задач была разработка утилиты, реализующей преобразование программ на РЕФАЛ-5 в соответствии с изменениями, внесенными в данный язык. Было решено представить утилиту в виде набора следующих модулей:

1. Лексический анализатор;
2. Синтаксического анализатора;
3. Семантический анализатор;
4. Преобразователь промежуточного представления программы в код на языке РЕФАЛ-5.

В рамках первых трёх из обозначенных выше модулей необходимо было произвести учёт изменений, внесенных в РЕФАЛ-5.

#### 3.1. Разработка лексического анализатора

Лексическим анализом называют процесс разбиения входной последовательности кодовых точек программы на токены в соответствии с лексической структурой заданного языка.

В ходе разработки лексического анализатора были выделены следующие лексические домены:

1. Идентификаторы;
2. Целые числа;
3. Переменные;
4. Левая вычислительная скобка вместе с именем функции;
5. Различные пунктуационные знаки: «(», «)», «{», «}», «>», «,», «:», «;», «=»;
6. Ключевое слово \$ENTRY;
7. Ключевое слово \$EXTERNAL и его разрешенные вариации: \$EXTRN, \$EXTERN;
8. Строковые комментарии;
9. Комментарии-вставки;

На этапе лексического анализа возможно учесть изменившуюся семантику двойных кавычек: поскольку в программах, написанных на устаревшей версии РЕФАЛа, подразумевается одинаковая обработка содержимого одинарных и двойных кавычек (как последовательности литер), необходимо заменить все вхождения двойных кавычек на одинарные. Такой подход позволяет сохранить логику, изначально заложенную в программу.

### 3.2 Разработка синтаксического анализатора

Синтаксический анализ подразумевает проверку входной последовательности токенов на соответствие грамматике рассматриваемого языка. Результатом выполнения синтаксического анализа может быть дерево разбора.

Для дальнейшего проведения семантического анализа было принято решение, что синтаксический анализатор должен порождать абстрактное синтаксическое дерево по грамматике, приведенной на Листинге 5.

На этапе синтаксического анализа было решено отслеживать в предложениях переменные, не содержащие точки после индикатора типа. Это невозможно совершить на этапе лексического анализа программы, поскольку переменные с опущенной точкой и названия функций относятся к одному домену и отличить их без контекстной информации нельзя.

### 3.3 Разработка семантического анализатора

Семантический анализ – это этап анализа программы, основной задачей которого является валидация семантической целостности программы.

На данном этапе было решено отслеживать ошибки регистра в именах функций при их вызове, а также замену вызова функции Туре на вызов функции Old-Туре, являющейся оберткой функции Туре и имитирующей её семантику. Данные действия невозможно совершить на предыдущих двух этапах, поскольку порядок объявлений функций в языке РЕФАЛ-5 произвольный и вызов функции в программе может встретиться раньше её объявления. Только после окончания синтаксического анализа, когда найдено определение функции и ее вызовы

выделены в отдельные структуры, становится возможно провести описанные здесь действия.

### 3.4 Разработка преобразователя промежуточного представления программы в код на языке РЕФАЛ-5

Заключительным этапом в трансляции программы с устаревшего стандарта языка РЕФАЛ-5 на актуальный является разбор абстрактного синтаксического дерева, построенного ранее, и построение по нему программы на РЕФАЛе-5. Грамматика абстрактного синтаксического дерева приведена на Листинге 5.

Листинг 5. Грамматика абстрактного синтаксического дерева

```
AST ::= Unit AST | empty
Unit ::= Function | Extern

Extern ::= ("Extern" Extern_decl)
Extern_decl ::= (Pos Name) Extern_decl1
Extern_decl1 ::= (Pos Name) Extern_decl1 | empty

Function ::= ("Function" (Pos) (Name) Scope Sentences)
Scope ::= "Entry" | "Local"

Sentences ::= Sentence Sentences1
Sentences1 ::= Sentence Sentences1 | empty
Sentence ::= ((Pattern) Conditions SentenceTail)

Conditions ::= Condition Conditions | empty
Condition ::= ("Condition" (Result) (Pattern))
SentenceTail ::= "RETURN" (Result) | "CALL-BLOCK" (Result) Sentences
Pattern ::= (Terms)
Result ::= (Terms)

Terms ::= Term Terms | empty
Term ::=
    ("Symbol Word" (Pos) Chars)
```

```
| ("Symbol Number" Number)
| ("Symbol Char" Char)
| ("Variable" (Pos) VarType Index)
| ("Brackets" Terms)
| ("Call" (Pos) (FunctionName) Terms)
VarType ::= 's' | 't' | 'e'
```

## 4 РЕАЛИЗАЦИЯ

Утилита, производящая преобразования программ в соответствии с изменениями, внесенными в язык РЕФАЛ-5, была реализована в том же виде, в каком была описана в предыдущем разделе. В дополнение к модулям анализа и преобразования программы был реализован небольшой модуль, осуществляющий координацию работы вышеобозначенных модулей, и оповещение пользователя о ходе работы над поданной на вход программой. Упрощенная схема работы утилиты приведена на Рисунке 3.

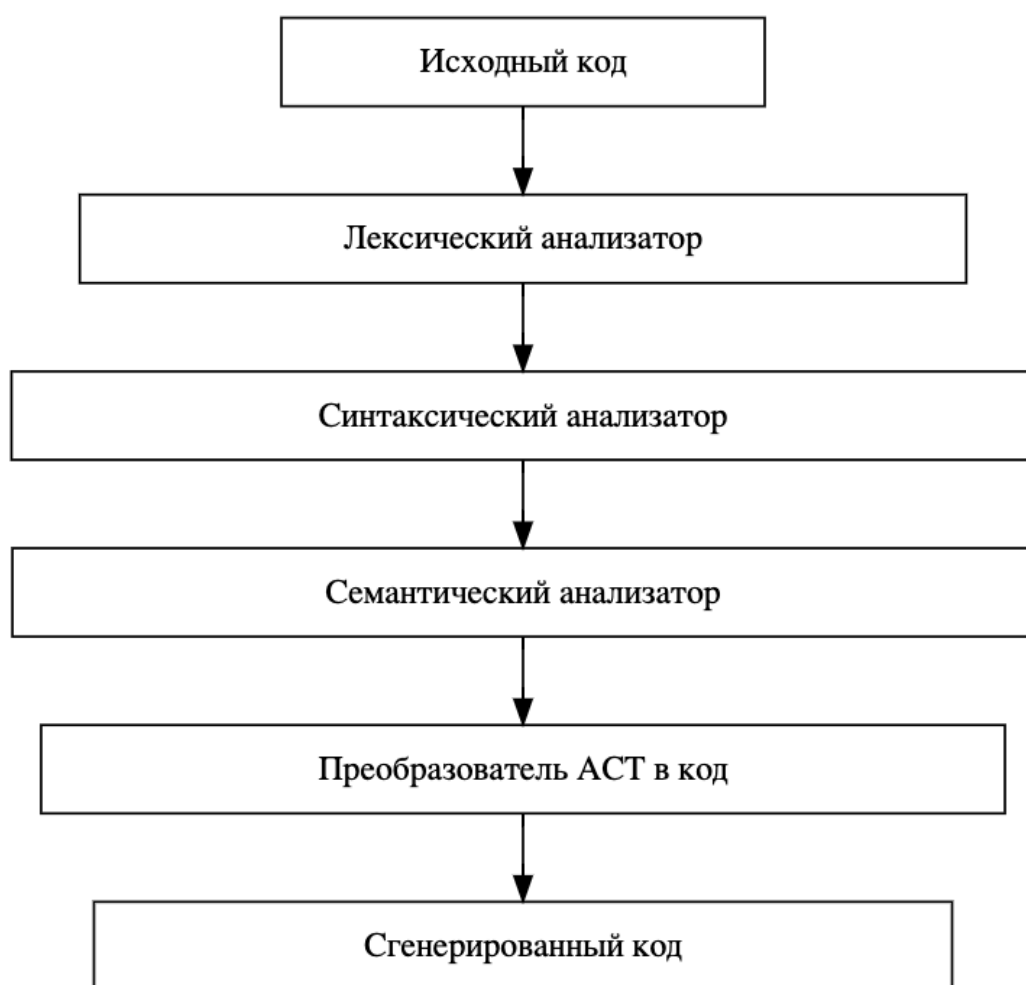


Рисунок 3. Схема работы утилиты

### 4.1 Реализация лексического анализатора

Для лексического анализа программы на языке РЕФАЛ-5 необходимо вызвать функцию <Scan e.FileName>. Предполагается, что на вход лексическому

анализатору подается имя файла, который необходимо преобразовать. Функция Scan прежде всего передает имя файла другим функциям, осуществляющим считывание кода из файла и его представления в виде набора строк, каждая из которых заключена в круглые скобки. Затем устанавливается значение текущей позиции (s.Line s.Col) равным (1 1), где s.Line – номер строки в тексте, а s.Col – номер столбца в строке, и преобразованный текст программы передается функции DoGetTokens

Функция DoGetTokens осуществляет анализ первого символа в текущей строке и в зависимости от того, какой символ был встречен, либо, если была выявлена односимвольная лексема, рекурсивно вызывает себя, либо передает управление другим функциям, осуществляющим разбор лексем, после чего вновь рекурсивно вызывает себя. Среди функций разбора лексем можно выделить:

1. GetKeyword - функция, производящая считывание всех символов, следующих за знаком «\$», вплоть до символа-разделителя. Если последовательность аккумулярованных символов, является ключевым словом, то в результате будет возвращен токен с соответствующим типом ключевого слова. В противном случае будет возвращен специальный токен, сигнализирующий об ошибке;
2. GetComment - функция, осуществляющая считывание содержимое комментария-вставки после символов «/\*». Данная функция собирает содержимое комментария. В случае нахождения окончания комментария в e.Line, все содержимое комментария извлекается и выделяется в соответствующий токен. Если в текущей строке окончания комментария найдено не было, то функция рекурсивно вызывает себя с целью дальнейшего разбора. В случае, если был достигнут конец файла, то будет возвращен токен с типом ошибка.
3. GetChars - функция, выделяющая последовательности литер, следующих за одинарными или двойными кавычками. Данная функция посимвольно просматривает текущую строку в поисках закрывающей кавычки нужного типа и аккумулирует содержимое

кавычек. В случае, если символы в строке закончились, а закрывающая кавычка не была найдена, возвращается токен с типом ошибка. В данной функции, кроме того, происходит учет изменившейся семантики двойных кавычек.

4. GetVar - функция, считывающая индекс переменной вплоть до символа-разделителя. Проверяет индекс в соответствии с установленными стандартами в языке РЕФАЛ. В случае обнаружения несоответствия возвращает токен-ошибку.
5. GetNumber - функция, производящая посимвольное считывание целого числа. У каждого следующего символа проверяется тип, и если он является десятичным числом, то считывание продолжается. Данная функция возвращает токен с типом число.
6. GetIdent - функция, выделяющая идентификаторы. Данная функция посимвольно просматривает текущую строку и аккумулирует значение. В случае встречи символа-разделителя и несоответствия аккумулированного содержимого домену идентификатора, возвращает токен с типом ошибка.

## 4.2 Реализация синтаксического анализатора

В связи с тем, что грамматика языка РЕФАЛ-5 представима в форме LL1, было принято решение проводить синтаксический анализ методом рекурсивного спуска. Каждому нетерминалу грамматики соответствует отдельная функция, реализующая соответствующее нетерминалу правило. Данный синтаксический анализатор не производит восстановление при ошибках и завершает работу с выводением сообщения об ошибке, в случае ее возникновения.

На вход синтаксическому анализатору подается последовательность токенов в виде скобочных термов. Функция CheckSyntax проверяет первый токен на соответствие его лексическим доменам ключевых слов и идентификатора и вызывает, соответственно, функции ParseExternDecl и ParseFunc.

Функция ParseExternDecl проверяет, относится ли первый токен к домену идентификатора и передает управление функции ParseExternalDeclNames,

которая уже рекурсивно продолжает считывать последовательность идентификаторов вплоть до окончания объявления внешней функции.

Функция `ParseFunc` производит проверку первого токена. Если он относится к домену идентификатор, то начинается анализ функции и вызывается функция `ParseBlock`.

Функция `ParseBlock` анализирует блок функции. Если первый токен обозначает открывающую фигурную скобку, то далее вызывается функция `ParseSentences` и после ее выполнения повторно проверяется наличие токена, соответствующего закрывающей фигурной скобке.

Функция `ParseSentences` производит проверку последовательности предложений. Она вызывает функцию `ParseSentence`, после чего сверяет наличие токена, обозначающего точку с запятой, и рекурсивно вызывает себя. Этот процесс продолжается до тех пор, пока не будет достигнута закрывающая фигурная скобка.

Функция `ParseSentence` разбивает предложение на левую и правую части в соответствии с грамматикой языка РЕФАЛ-5 и вызывает функции для их разбора: `ParseExpression` и `ParseRightSide`.

Функция `ParseExpression` производит анализ последовательности токенов, пока не встретит токен, который не может быть составной частью выражения или образца, что зависит от символа-флага, который данная функция принимает на вход. Кроме того, в рамках данной функции токены-идентификаторы, соответствующие устаревшему обозначению переменных без точки, заменяются на токены-переменные.

Функция `ParseRightSide` определяет наличие условных конструкций в правой части предложения и в зависимости от ситуации вызывает функцию `ParseExpression` или `ParseConditions`.

Функция `ParseConditions` сначала вызывает функцию `ParseExpression`, чтобы отделить содержимое выражения, затем проверяет наличие токена, обозначающего точку с запятой, после чего вызывает функцию `ParseConditionTail`.

Функция `ParseConditionTail` анализирует, какое окончание имеет условная конструкция. Если первый токен соответствует фигурной скобке, значит, обнаружен блок и управление передается функции `ParseBlock`. В противном случае снова вызывается функция `ParseExpression`, которая выделит содержимое образца, после чего будет вызвана функция `ParseRightSide`.

### 4.3 Реализация семантического анализатора

Результатом успешного проведения синтаксического анализа является абстрактное синтаксическое дерево. Не будем останавливаться на определении каждой функции, но опишем проведение семантического анализа в целом.

На этапе семантического анализа сначала производится обход объявлений функций: как внешних, так и внутренних, с целью сбора имен всех функций для последующей проверки вызовов функций на корректность. После этого проводится проверка предложений в составе функций: для каждого предложения собирается набор переменных, присутствующих в образцах и производится сверка переменных и вызовов функций в правых частях предложений.

В случае, если в выражении найдена переменная или вызов функции, которая не была до этого объявлена, то возвращается сообщение об ошибке.

Если найден вызов функции и при этом ее имя использовано в неверном регистре, то возвращается особое скобочное выражение, сигнализирующее о необходимости замены в данном вызове функции имеющееся имя на корректное.

После завершения обхода всего дерева проводится анализ значений, возвращенных вызванными функциями. Возможны следующие варианты:

1. Не было ничего возвращено. Подобный результат говорит об успешном завершении семантического анализа и отсутствии необходимости проводить дополнительные преобразования.
2. Был возвращен непустой результат. В данном случае необходимо отделить особые скобочные выражения, описанные выше, от других сообщений об ошибках и провести точечные правки в абстрактном синтаксическом дереве. Если после сортировки выражений не было обнаружено ни одного сообщения об ошибке, то семантический анализ завершился успешно со внесением некоторых корректив. В

противном случае в анализируемой программе присутствуют семантические ошибки. Анализ завершается сообщением о неудаче.

#### 4.4 Реализация преобразователя промежуточного представления программы в код на языке РЕФАЛ-5

После прохождения программой всех фаз анализа, проведения над ней необходимых преобразований и внесения различных правок, производится последний обход абстрактного синтаксического дерева с целью генерации модифицированной программы. Исходное форматирование программы не сохраняется, так как символы-разделители на этапе лексического анализа игнорируются.

## 5 ТЕСТИРОВАНИЕ

После обновления SCP3 с помощью реализованной в ходе данной ВКР утилиты необходимо было проверить работоспособность данного суперкомпилятора. Для решения поставленной задачи был проведен ряд тестов.

В данном разделе описано три различных теста, а также произведено сравнение результатов их проведения для оригинальной и обновленной версий суперкомпилятора SCP3 .

### 5.1 Суперкомпиляция ленивого интерпретатора

Плоский РЕФАЛ не позволяет выполнять ленивые вычисления. Чтобы обойти это ограничение, в ходе разработки SCP3 на плоском РЕФАЛе был реализован ленивый интерпретатор Lazy-Int полного функционального языка (ограниченного РЕФАЛа). Некоторая программа, написанная на ограниченном РЕФАЛе, интерпретируется с помощью Lazy-Int и суперкомпилируется с помощью SCP3 согласно MST-схеме, приведенной на Листинге 6.

Листинг 6. MST-схема для теста с ленивыми вычислениями.

```
< Scp . . . . . >
  < Lazy-Int . . . . e.1 . >
    < Fbc < Fab ! > >
```

Программа, неявно присутствующая в вызове Lazy-Int, определяет две функции: Fab и Fbc. Fab – неплоская рекурсивная функция, производящая замену префикса, состоящего из некоторой последовательности символов «a» на префикс той же длины из символов «b». Данная программа представлена на Листинге 7.

Листинг 7. Функция Fab

```
Fab {
  'a' e.1 = 'b' <Fab e.1>;
  e.1 = e.1 ;
}
```

Функция Fbc определена аналогично и производит замену префикса, состоящего из некоторой последовательности символов «b» на префикс той же длины, но уже содержащий символы «c». Ленивый интерпретатор работает с композицией этих двух функций.

Полученная в результате выполнения теста с актуализированным суперкомпилятором SCP3 программа оказалась идентична остаточной программе, полученной с помощью старой версии суперкомпилятора, запущенным под DOSBox. Как видно из приведенного на Листинге 8 кода остаточной программы, результат её выполнения будет идентичен вызову композиции функций <Fbc<Fab>>.

Листинг 8. Функция FAC

```
FAC {
  ('a'e.1 ) = <F1C1 () (e.1 )> ;
  ('bb'e.1 ) = <F1C2 () (e.1 )> ;
  ('b'e.1 ) = 'c'e.1 ;
  (e.1 ) = e.1 ;
}
F1C1 {
  (e.27 ) ('a'e.28 ) = <F1C1 (e.27 'c') (e.28 )> ;
  (e.27 ) ('b'e.28 ) = <F1C2 (e.27 ) (e.28 )> ;
  (e.27 ) (e.28 ) = 'c'e.27 e.28 ;
}
F1C2 {
  (e.27 ) ('b'e.28 ) = <F1C2 (e.27 'c') (e.28 )> ;
  (e.27 ) (e.28 ) = 'c'e.27 'c'e.28 ;
}
```

Изменим функцию Fab таким образом, чтобы она заменяла все вхождения символа «a» на «b». Код измененной функции приведен на Листинге 9.

Внесем в функцию Fbc аналогичные изменения и проведем суперкомпиляцию. Результаты использования старого и актуализированного

суперкомпиляторов вновь оказались идентичны. Остаточная программа, полученная в результате суперкомпиляции, приведена на Листинге 10.

#### Листинг 9. Измененная функция Fab

```
Fab {  
    'a' e.1 = 'b' <Fab e.1>;  
    s.2 e.1 = s.2 <Fab e.1>;  
    = ;  
}
```

#### Листинг 10. Функция FAC

```
FAC {  
    ('a'e.1 ) 'a' = <F1C1 () (e.1 )> ;  
    ('b'e.1 ) 'b' = <F1C3 () (e.1 )> ;  
    (s.21 e.1 ) s.21 = <F1C6 s.21 () (e.1 )> ;  
    () s.21 = ;  
}  
F1C1 {  
    (e.27 ) ('a'e.28 ) = <F1C1 (e.27 'c') (e.28 )> ;  
    (e.27 ) ('b'e.28 ) = <F1C3 (e.27 'c') (e.28 )> ;  
    (e.27 ) (s.31 e.28 ) = <F1C3 (e.27 s.31 ) (e.28 )> ;  
    (e.27 ) () = 'c'e.27 ;  
}  
F1C3 {  
    (e.44 ) ('a'e.45 ) = <F1C1 (e.44 'c') (e.45 )> ;  
    (e.44 ) ('b'e.45 ) = <F1C3 (e.44 'c') (e.45 )> ;  
    (e.44 ) (s.48 e.45 ) = <F1C3 (e.44 s.48 ) (e.45 )> ;  
    (e.44 ) () = 'c'e.44 ;  
}  
F1C6 {  
    s.21 (e.165 ) ('a'e.166 ) = <F1C5 s.21 (e.165 ) (e.166 )> ;  
    s.21 (e.165 ) ('b'e.166 ) = <F1C6 s.21 (e.165 'c') (e.166 )> ;  
    s.21 (e.165 ) (s.169 e.166 )  
        = <F1C6 s.21 (e.165 s.169 ) (e.166 )> ;  
    s.21 (e.165 ) () = s.21 e.165 ;  
}
```

```

F1C5 {
    s.21 (e.165 ) ('a'e.166 ) = <F1C5 s.21 (e.165 'c')(e.166 )> ;
    s.21 (e.165 ) ('b'e.166 ) = <F1C6 s.21 (e.165 'cc')(e.166 )>
;
    s.21 (e.165 ) (s.172 e.166 )
        = <F1C6 s.21 (e.165 'c's.172 ) (e.166 )> ;
    s.21 (e.165 ) () = s.21 e.165 'c';
}

```

## 5.2 Суперкомпиляция интерпретатора

В данном тесте использовался интерпретатор графов Int. Программа Search, написанная на Плоском РЕФАЛе, интерпретируется с помощью Int и суперкомпилируется с помощью SCP3 согласно MST-схеме, приведенной на Листинге 11.

Листинг 11. MST-схема для теста

```

<Scp ..... >
  <Int .... s.U s.V e.W .. >
    <Search ( ! ! ) ! >

```

Функция Search является метакодированным аргументом функции Int. Search – плоская рекурсивная функция, выполняющая роль предиката: она возвращает значение TRUE, если два символа, заключенные в скобки, идентичны двум другим символам, следующим за скобками, а в противном случае FALSE. Код функции Search приведен на Листинге 12.

Листинг 12. Функция Search

```

Search {
    (s.X s.Y) s.X s.Y e.S = TRUE;
    (s.X s.Y) s.Z e.S = <Search (s.X s.Y) e.S>;
    (s.X s.Y) = FALSE;
}

```

В данном тесте суперкомпилятор производит классификацию (s.U s.V) образцов. Остаточные программы, полученные при проведении теста со старым и обновленным суперкомпилятором идентичны. Код программы приведен на Листинге 13.

### Листинг 13. Функция SEARCH

```
SEARCH {
    (e.3 ) s.2 s.2  = <F1C1 (e.3 )s.2 > ;
    (e.3 ) s.2 s.1  = <F1C3 (e.3 )s.1 s.2 > ;
}
F1C1  {
    (s.27 s.27 e.3 ) s.27  = TRUE ;
    (s.25 s.27 e.3 ) s.25  = <F1C1 (e.3 )s.25 > ;
    (s.25 ) s.25  = FALSE ;
    (s.25 e.3 ) s.2  = <F1C1 (e.3 )s.2 > ;
    () s.2  = FALSE ;
}
F1C3  {
    (s.47 e.3 ) s.47 s.2  = <F1C2 s.47 s.2 (e.3 )> ;
    (s.47 e.3 ) s.1 s.47  = <F1C3 (e.3 )s.1 s.47 > ;
    (s.47 e.3 ) s.1 s.2  = <F1C3 (e.3 )s.1 s.2 > ;
    () s.1 s.2  = FALSE ;
}
F1C2  {
    s.47 s.49 (s.49 e.3 )  = TRUE ;
    s.49 s.2 (s.49 e.3 )  = <F1C2 s.49 s.2 (e.3 )> ;
    s.47 s.2 (s.49 e.3 )  = <F1C3 (e.3 )s.47 s.2 > ;
    s.47 s.2 ()  = FALSE ;
}
```

## 6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

В данном разделе описываются способы установки и использования утилиты, реализованной в ходе данной ВКР, а также приведены инструкции для суперкомпилятора SCP3.

Для работы утилиты, осуществляющей трансформацию программы в соответствии с изменениями, внесенными в синтаксис языка РЕФАЛ-5, необходимо следующее программное обеспечение:

1. Операционная система GNU/Linux, MacOS или Windows;
2. Компилятор языка РЕФАЛ-5 версии 2000 года или новее или компилятор языка РЕФАЛ-5λ [6] версии 2.3.2.

### 6.1 Установка утилиты и запуск

Для работы с утилитой необходимо скомпилировать её исходные файлы:

1. Translator.ref;
2. Lexer.ref;
3. Parser.ref;
4. Plainer.ref;
5. FileWorker.ref;
6. AST.ref;
7. Tokens.ref;

На Листинге 14 представлена команда для установки утилиты с использованием компилятора языка РЕФАЛ-5λ `srefc`. При успешном выполнении этой команды будет произведена сборка утилиты, сгенерирован исполняемый файл `Translator`, а также следующие временные файлы:

1. Translator.rasl;
2. Lexer.rasl;
3. Parser.rasl;
4. Plainer.rasl;
5. FileWorker.rasl;
6. AST.rasl;
7. Tokens.rasl.

Для того, чтобы избавиться от них, достаточно выполнить команду, приведенную на Листинге 17 на системе Windows или команду, указанную на Листинге 18 на unix-like системах.

#### Листинг 14. Команда для компиляции исходных файлов утилиты

```
srefc Translator Plainer Parser Lexer AST Tokens
```

На Листинге 15 приведена команда для установки утилиты с использованием компилятора языка РЕФАЛ-5 `refc`. В случае, если выполнение команды прошло успешно, будут сгенерированы \*.rsl файлы.

#### Листинг 15. Команда для компиляции исходных файлов утилиты

```
refc Translator Plainer Parser Lexer AST Tokens
```

Для запуска утилиты с помощью интерпретатора `refgo` необходимо выполнить команду, приведенную на Листинге 16. Для запуска утилиты, скомпилированной `srefc`, достаточно запустить сгенерированный на предыдущем этапе исполняемый файл `Translator`.

#### Листинг 16. Команда для запуска утилиты

```
refgo Translator+Plainer+Parser+Lexer+AST+Tokens
```

#### Листинг 17. Команда для удаления файлов \*.rsl на системе Windows

```
del *.rsl
```

#### Листинг 18. Команда для удаления файлов \*.rsl на unix-line системах

```
rm *.rsl
```

## 6.2 Использование утилиты

Работа с утилитой осуществляется посредством командной строки. В зависимости от выбранного компилятора способ запуска утилиты будет

различаться. Если программа была установлена с использованием компилятора РЕФАЛ-5, то для её запуска необходимо использовать интерпретатор refgo. В противном случае достаточно запустить исполняемый файл, полученный ранее.

После запуска утилиты пользователю будет предложено ввести два аргумента: имя файла, который необходимо преобразовать, и имя файла, в который будет записан результат в случае успешного выполнения программы. Если на вход утилите был подан лишь один аргумент, либо выполнить преобразование программы, поданной на вход, невозможно, будет выведено сообщение об ошибке и утилита завершит работу. Предусмотрены следующие сообщения об ошибках:

1. Сообщение о некорректном количестве аргументов, поданных на вход утилите;
2. Сообщение об ошибке, возникшей на этапе лексического анализа и сигнализирующей об обнаружении в тексте программы лексемы, не относящейся ни к какому лексическому домену из заданных;
3. Сообщение о синтаксической ошибке, заключающейся в несоответствии структуры программы грамматике языка РЕФАЛ-5.
4. Сообщение об ошибке в семантике программы.

Пример сообщения об ошибке приведен на Листинге 19.

#### Листинг 19. Пример сообщения об ошибке

```
*Enter source and dest files names
example.ref result.ref
*Lexical analysis completed
Starting syntax and semantic analysis
((7 15 example.ref) variable e.12 not declared)
```

Каждый успешно выполненный этап работы сопровождается сообщением, сигнализирующем о завершении предыдущего этапа и перехода к следующему. В случае успешного проведения всех преобразований будет выведено соответствующее сообщение. Пример запуска утилиты приведен на Листинге 20.

#### Листинг 20. Пример нормального завершения работы утилиты

```
*Enter source and dest files names
example.ref result.ref
*Lexical analysis completed
Starting syntax and semantic analysis
*Syntax and semantic analysis completed
*****SUCCESS*****
```

### 6.3 Использование суперкомпилятора SCP3

Суперкомпилятор SCP3 не имеет графического интерфейса. Работа с ним производится с помощью командной строки. SCP3 принимает на вход задание на суперкомпиляцию в виде файла формата .mst и файл с исходным кодом программы на Плоском РЕФАЛе, являющейся объектом суперкомпиляции.

Для обработки задания на суперкомпиляцию и ее выполнения на Windows необходимо использовать скрипты mst.bat и scp3.bat. Для unix-like систем предусмотрены аналогичные скрипты mst.sh и scp3.sh.

Для генерации остаточной программы по построенному суперкомпилятором графу необходимо выполнить скрипт cgr5.bat или cgr5.sh соответственно.

Примеры использования суперкомпилятора на unix-like системах продемонстрированы на Листингах 21, 22 и 23.

#### Листинг 21. Команда для обработки задания на суперкомпиляцию

```
./mst file_name.mst
```

#### Листинг 22. Команда для выполнения суперкомпиляции

```
./scp3 file_name
```

#### Листинг 23. Команда для генерации остаточной программы

```
./cgr5 file_to_write_in
```

## ЗАКЛЮЧЕНИЕ

В ходе данной выпускной квалификационной работы было произведено ознакомление с языком РЕФАЛ-5 и изучены принципы работы суперкомпилятора SCP3.

Была разработана утилита, реализующая преобразование программ на РЕФАЛе-5 в соответствии с изменениями, внесенными в данный язык программирования. Также с помощью данной утилиты перенесена под актуальную версию языка РЕФАЛ-5 реализация суперкомпилятора SCP3.

Было проведено тестирование актуализированного суперкомпилятора SCP3, в результате которого была установлена работоспособность данного суперкомпилятора.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. В. Ф. Турчин, “Метаязык для формального описания алгоритмических языков”, Цифровая вычислительная техника и программирование, Советское радио, Москва, 1966, 116–119.
2. Turchin V. F. Refal-5: programming guide and reference manual. 1999, (Revised and extended edition of the issue published by New England Publishing Co., Holyoke 1989)
3. Turchin V., Nemytykh A. Metavariables: Their implementation and use in Program transformation, CCNY Technical Report CSc TR-95-012, 1995
4. Nemytykh A.P., Pinchuk V.A., Turchin V.F. (1996) A Self-Applicable supercompiler. In: Danvy O., Glück R., Thiemann P. (eds) Partial Evaluation. Lecture Notes in Computer Science, vol 1110. Springer, Berlin, Heidelberg
5. Климов А. В., Романенко С. А. Суперкомпиляция: основные принципы и базовые понятия // Препринты ИПМ им. М. В. Келдыша. — 2018. — No111. — 36 с.
6. Компилятор Рефал-5 $\lambda$ [Электронный ресурс] – Режим доступа: <https://github.com/bmstu-iu9/refal-5-lambda>
7. Немытых А. П. О некоторых понятиях суперкомпиляции – методе специализации программ // Сборник трудов по функциональному языку программирования Рефал, том II // Под редакцией А. П. Немытых. — Переславль-Залесский: Издательство «СБОРНИК», 2015. — 156 с.
8. Немытых А.П. О суперкомпиляции // [Электронный ресурс]. — Доступ: [http://conf.nsc.ru/files/conferences/Lyap100/fulltext/69293/69928/nemytykh\\_supercompilation\\_Lyapunov100.pdf](http://conf.nsc.ru/files/conferences/Lyap100/fulltext/69293/69928/nemytykh_supercompilation_Lyapunov100.pdf)
9. Turchin V.: Metacomputation Metasystem transitions plus supercompilation, LNCS vol. 1110 (1996)
10. Turchin V. F., The Phenomenon of Science, Columbia University Press, 1977
11. Turchin V.F. The Language Refal, the Theory of Compilation and Metasystem Analysis, Courant Computer Science Report #20, New York University, 1980

## ПРИЛОЖЕНИЕ А

### Листинг 24. Грамматика языка РЕФАЛ-5

```
program ::= f-definition program
          | external-decl ';' program
          | empty

f-definition ::= f-name block semicolon
              | $ENTRY f-name block semicolon

semicolon ::= ';' | empty

external-decl ::= $EXTERNAL f-name-list
                | $EXTERN f-name-list
                | $EXTRN f-name-list

f-name-list ::= f-name f-name-list1

f-name-list1 ::= ',' f-name-list | empty

block ::= { sentences }

sentences ::= sentence sentences1

sentences1 ::= ';' sentences | empty

sentence ::= pattern right-side

right-side ::= '=' expression
            | conditions

conditions ::= ',' expression ':' cond-end

cond-end ::= pattern right-side
           | block
```

```
expression ::= empty
             | term expression

term ::= symbol
      | variable
      | (expression)
      | <f-name expression>

f-name ::= identifier
```