

«Московский государственный технический
университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ *ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ*
КАФЕДРА *ТЕОРЕТИЧЕСКАЯ ИНФОРМАТИКА*
 И КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту на тему:
Оптимизация образцовых выражений в языке
программирования РЕФАЛ

Студент

Батусов П. В.

Руководитель курсового проекта

Коновалов А. В.

Аннотация

В данной работе рассматриваются возможные методы и подходы для оптимизации образцовых выражений в языке программирования РЕФАЛ. Цель поставленной задачи — доработка компилятора, порождающего более быструю исполняемую программу. В работе описывается и обосновывается необходимость данной оптимизации, а также производится оценка производительности оптимизированных программ.

Содержание

ВВЕДЕНИЕ	3
1 Базисный РЕФАЛ	4
1.1 Предложения и функции	4
1.2 Сопоставление с образцом	5
1.3 Необходимость оптимизации	6
1.4 Постановка задачи	6
2 Алгоритм оптимизации	7
2.1 Жесткий образец	7
2.2 Уточнение образцов	8
2.3 Обобщение образцов	9
2.4 Сложнейший жесткий образец	10
2.5 Классы образцов	11
2.6 Оптимизация	12
3 Реализация алгоритма	14
3.1 Внутренне представление компилируемой программы	14
3.1.1 Лексический анализ	14
3.1.2 Синтаксический анализ	14
3.1.3 Пример синтаксического дерева	14
3.2 Обобщение образцов	15
3.2.1 Генерация жестких образцов	15
3.2.2 Быстрое обобщение	16
ЗАКЛЮЧЕНИЕ	17

ВВЕДЕНИЕ

РЕФАЛ — РЕкурсивных Функций АЛгоритмический язык программирования, является одним из старейших функциональных языков, ориентированный на символьные вычисления: обработку символьных строк (например, алгебраические выкладки); перевод с одного языка (искусственного или естественного) на другой; решение проблем, связанных с искусственным интеллектом. Соединяет в себе математическую простоту с практической ориентацией на написание больших и сложных программ [1].

Первая версия РЕФАЛа была создана в 1966 году Валентином Турчиным в качестве метаязыка для описания семантики других языков. Впоследствии, в результате появления достаточно эффективных реализаций на ЭВМ, он стал находить практическое использование в качестве языка программирования [2].

РЕФАЛ программа может состоять из одного или нескольких модулей (файлов), каждый из которых, в свою очередь, состоит из функций. РЕФАЛ-функция представляет собой упорядоченный набор предложений, состоящих из образца и шаблона. На вход функции подается некоторое выражение; вычисление функции состоит в сопоставлении выражения поочередно образцам, взятым из первого, второго и т. д. предложений. Если очередное сопоставление проходит успешно, то на основании шаблона, взятого из того же предложения, формируется новое РЕФАЛ-выражение, которое и будет результатом функции. В случае, если ни с одним из имеющихся образцов аргумент функции сопоставить не удалось, фиксируется ошибка (аварийно завершается вся программа). Во избежание этого часто в конце функции помещают предложение, с образцом которого можно сопоставить вообще произвольное выражение. В некоторых современных реализациях РЕФАЛа (например, РЕФАЛ+) неуспех любого выражения функции вместо ошибки порождает неуспех самой функции.

В РЕФАЛе, как и во многих других функциональных языках программирования, в отличие от императивных языков, отсутствуют циклы. Для описания циклических процессов используется рекурсия. Зачастую, при таком подходе, тело функции, реализующей цикл, содержит большое число предложений, с очень похожими образцами. Если оптимизировать их вычисление, например создать дерево из левых частей предложений (в пределах одной функции), чтобы операторы, общие для нескольких предложений, выполнялись только один раз, это может дать большой прирост скорости выполнения программы.

1 Базисный РЕФАЛ

РЕФАЛ является одним из старейших функциональных языков, отличительная черта которого — использование сопоставления с образцом и переписывания термов как основного способа определения функций.

1.1 Предложения и функции

Предложением называется конструкция вида: «образцовое выражение = результатное выражение;». Образец часто называют левой частью, результат — правой частью, в которой могут использоваться только те переменные, которые определены в левой части.

РЕФАЛ-функция представляет собой упорядоченный набор таких предложений. Ее выполнение заключается в сопоставлении переданного аргумента с левой частью первого предложения. Если сопоставление оказывается успешным, то значения переменных, определенных в образце, подставляются в правую часть и вызов функции заменяется на получившийся результат. При неудачном сопоставлении с образцом, точно таким же образом выполняется второе предложение и т. д. Если сопоставление не удалось в левой части последнего предложения, то программа прекращает работу с ошибкой невозможности сопоставления (recognition impossible).

Обычно в левых частях выражений рассматриваются различные варианты допустимых аргументов, поэтому ошибка невозможности сопоставления часто говорит о том, что или на вход функции был передан некорректный аргумент, или сама функция написана с ошибкой. Таким образом, в РЕФАЛ четко разделены операции анализа аргумента, выполняемые при помощи сопоставления с образцом, и операции синтеза, осуществляемые при помощи подстановки значений переменных в результатное выражение.

В качестве аргументов функции передается последовательность (возможно пустая) термов, которыми могут быть:

- обыкновенные символы — буквы, цифры и т. д.;
- символы-метки (идентификаторы);
- цифры — цифровая запись неотрицательных целых чисел, не превышающих предельное значение;
- выражение в структурных скобках;

— активное выражение (означающее вызов функции);

1.2 Сопоставление с образцом

Образцы РЕФАЛа могут содержать свободные переменные, которые состоят из указателя типа, точки и индекса. Индексом переменной является последовательность букв и цифр. Есть три указателя типа. Они обозначаются малыми буквами *s*, *t* и *e*, а соответствующие им переменные называются *s*-, *t*- и *e*-переменными. Различие между ними состоит в множествах их допустимых значений. Значением *s*-переменной может служить только один символ, а *t*-переменной — любой терм. Значением *e*-переменной может быть любое выражение [3].

Если в образце присутствует несколько вхождений переменных с одинаковым типом и одинаковым именем, то такие переменные называются повторными. Все вхождения повторной переменной должны иметь одинаковое значение. В образце не может быть переменных, имеющих одинаковое имя, но при этом разные типы.

Сопоставлением объектного выражения *E* с образцом *P* называется поиск значений переменных, входящих в образец *P*, подстановка которых в *P* дает выражение *E*. Если такие значения найти невозможно, то сопоставление считается неуспешным.

Если существует несколько вариантов сопоставления, то выбирается тот, в котором первая переменная (при чтении слева-направо) принимает кратчайшее значение (имеется ввиду длина в термах). Если это правило не устраняет неоднозначности, то анализируется следующая переменная и т. д. При таком соглашении сопоставление в РЕФАЛе становится однозначной операцией.

Для разных образцовых выражений операция сопоставления будет выполняться за разное время. Так, например простые операции: сопоставление с атомом, *s*-переменной, с парой скобок выполняются за постоянное время, а сопоставления каждой открытой *e*-переменной линейно зависят от длины «сканируемого» объектного выражения, поэтому одна *e*-переменная добавляет линейную сложность, две — квадратичную, три — кубическую. Повторная *t*- или *e*-переменная требует рекурсивного сравнения объектных термов или объектных выражений, соответственно, поэтому сложность сопоставления линейно зависит от числа атомов и скобок, входящих в сравниваемые выражения. Пара повторных переменных требует одного сравнения, 3 переменных — двух сравнений и т. д.

1.3 Необходимость оптимизации

Синтаксис РЕФАЛа не позволяет функциям принимать более одного аргумента — объектного выражения. При передаче нескольких параметров, они «склеиваются» в одно объектное выражение, которое потом в теле функции приходится разбирать на части соответствующие входным аргументам. Таким образом зачастую можно встретить ситуацию, когда несколько образцовых частей предложения одной функции имеют похожую структуру, в результате чего одни и те же операции анализа аргумента функции выполняются несколько раз. Это крайне неэффективно, особенно если такая функция описывает некоторый циклический процесс. Кроме этого, оптимизация необходима в случае если компилятор РЕФАЛа порождает код для каждого предложения функции отдельно (пример — рис. 1).

Код на Рефале	Код на C++
<pre>// Объявления библиотечных функций \$EXTERN WriteLine, Dec, Mul; // Объявление локальной функции \$FORWARD Fact; // Точка входа в программу \$ENTRY Go { = <WriteLine '6! = ' <Fact 6>>; }</pre>	<pre>// Automatically generated file. Don't edit! #include "refalrts.h" extern refalrts::FnResult WriteLine(refalrts::Iter arg_begin, refalrts::Iter arg_end); extern refalrts::FnResult Dec(refalrts::Iter arg_begin, refalrts::Iter arg_end); extern refalrts::FnResult Mul(refalrts::Iter arg_begin, refalrts::Iter arg_end); static refalrts::FnResult Fact(refalrts::Iter arg_begin, refalrts::Iter arg_end); refalrts::FnResult Go(refalrts::Iter arg_begin, refalrts::Iter arg_end) { Код предложения return refalrts::cRecognitionImpossible; } static refalrts::FnResult Fact(refalrts::Iter arg_begin, refalrts::Iter arg_end) { Код первого предложения Код второго предложения return refalrts::cRecognitionImpossible; }</pre>

Рис. 1 – Генерация кода в простом РЕФАЛе

1.4 Постановка задачи

Задачей данного курсового проекта является разработка алгоритма оптимизации, устраняющего избыточность операции анализа аргумента функции для предложений, имеющих похожую структуру образцовых частей.

2 Алгоритм оптимизации

Одним из возможных методов удаления избыточных сопоставлений с образцом является обобщение левых частей подряд идущих предложений.

2.1 Жесткий образец

Введем некоторые определения. *Объектное выражение* (или, для краткости, просто «выражение») — это последовательность из термов, каждый из которых может быть либо атомарным, либо скобочным термом, т.е. выражением, заключенным в круглые скобки. *Атомарные термы* (атомы) — это термы, которые невозможно разбить на составные части путем сопоставления с образцом, обычно к ним относятся атомы-числа, атомы-символы, атомы-имена и т.д. Будем обозначать атомы как X_1, X_2, \dots, X_n . Сами объектные выражения будем обозначать латинской буквой E , иногда с индексом.

Образцовое выражение (образец) — это синтаксическая конструкция, позволяющая описать некоторое множество объектных выражений. Оно записывается как объектное выражение, но некоторые части заменяются особыми подстановочными знаками — переменными. Переменная может заменять либо любой атом (так называемая s-переменная), либо любой терм (t-переменная), либо некоторый правильный (с правильной расстановкой скобок) фрагмент выражения (e-переменная). Переменные могут иметь имена для ссылок на них из результатных выражений, либо для указания того факта, что две одноименные переменные (они должны быть одного типа) заменяют одну и ту же часть объектного выражения.

Жесткий образец — выражение без открытых и повторных переменных. Таким образом жесткое выражение на каждом уровне скобок содержит не более чем одну e-переменную, а индексы всех переменных, входящих в одно и то же жесткое выражение, должны быть попарно различны. Таким образом, жесткие образцы имеют следующий вид:

Образец := Терм* [e-переменная Терм*].

Терм := Атом | t-переменная | s-переменная | (Образец).

Атом := X | s-переменная.

2.2 Уточнение образцов

Подстановка — замкнутая на множестве жестких образцов замена переменных в образце соответствующими образцами ($S = \{e_1 \rightarrow p_1, t_2 \rightarrow p_2, \dots\} P \xrightarrow{S} P'$). Один из возможных примеров подстановки (1).

$$\begin{aligned} P &= t_1 e_2 t_3 \\ S &= \{t_1 \rightarrow (e_4), e_2 \rightarrow s_5 e_6\} \\ P' &= (e_4) s_5 e_6 t_3 \end{aligned} \tag{1}$$

Если один образец P_1 можно получить из другого образца P_2 подстановкой, то будем говорить, что P_1 *уточняет* P_2 , а P_2 , соответственно, *обобщает* P_1 . Введем следующие обозначения для операции уточнения образцов:

$P_1 \Rightarrow^+ P_2$ — P_1 уточняет P_2

$P_1 \Rightarrow^* P_2$ — нестрогое уточнение, $P_1 \Rightarrow^+ P_2$ или $P_1 = P_2$

Не трудно догадаться, что e обобщает любой образец: $\forall P e \Rightarrow^* P$.

Пусть $P_1 \Rightarrow^+ P_2$, и при этом не существует такого образца P_3 , что $P_1 \Rightarrow^+ P_3 \Rightarrow^+ P_2$. Тогда P_1 будем называть *минимальным уточнением* P_2 (и, соответственно, P_2 — *минимальным обобщением* P_1) и обозначать как $P_1 \Rightarrow_{\min} P_2$.

Заметим, что $P_1 \Rightarrow_{\min} P_2$ тогда и только тогда, когда P_1 можно получить из P_2 только одной из следующих замен:

$$\begin{aligned} e &\rightarrow te \\ e &\rightarrow et \\ t &\rightarrow (e) \\ t &\rightarrow s \\ s &\rightarrow X \end{aligned} \tag{2}$$

Если $P \Rightarrow^+ Q$ и можно построить цепочку минимальных уточнений от Q до P двумя способами, т. е.

$$\begin{aligned} P &\Rightarrow_{\min} R_1 \Rightarrow_{\min} R_2 \Rightarrow_{\min} \dots \Rightarrow_{\min} R_n \Rightarrow_{\min} Q \\ P &\Rightarrow_{\min} R'_1 \Rightarrow_{\min} R'_2 \Rightarrow_{\min} \dots \Rightarrow_{\min} R'_m \Rightarrow_{\min} Q \end{aligned}$$

то $n = m$.

Чтобы это показать, ведём числовую характеристику образца $C(P)$, которую определим как

$$C(P) = n_t + 2n_s + 3n_X + 3n_{()} - n_e + 1$$

Здесь n_t, n_s, n_e — число, соответственно t -переменных, s -переменных, e -переменных, $n_()$ — число пар круглых скобок, n_X — число литералов атомов. Нетрудно убедиться, что для каждого преобразования (2) величина $C(P)$ возрастает на 1, следовательно, минимальное уточнение увеличивает данную величину на 1. Поскольку $C(P)$ и $C(Q)$ зависят только от внешнего вида образца, число элементарных уточнений между P и Q не будет зависеть от «траектории» перехода. Определим *сложность сопоставления* для образца P как число минимальных уточнений от e до $P - C(P)$. Заметим, что $C(e) = 0$.

2.3 Обобщение образцов

Рассмотрим множество образцов $P_1 \dots P_N$. Будем называть *глобальным сложнейшим обобщением (ГСО)* такое $P^* \Rightarrow^* P_i, i = 1 \dots N$, что $\nexists Q \Rightarrow^* P_i, C(Q) > C(P^*)$. Аналогичным образом определим *локально сложнейшее обобщение (ЛСО)* $P^* \Rightarrow^+ Q$. Заметим, что $\text{ГСО}(P_1 \dots P_N) \subseteq \text{ЛСО}(P_1 \dots P_N)$. Ниже (3), представлен пример ЛСО и ГСО для двух образцов $P_1 = stt$ и $P_2 = st$.

$$\begin{aligned} \text{ЛСО}(stt, st) &= \{ste, set, ett\} \\ \text{ГСО}(stt, st) &= \{ste, set\} \\ e_1 t_2 t_3 &\xrightarrow{e_1 \rightarrow s} st_2 t_3 \\ e_1 t_2 t_3 &\xrightarrow{e_1 \rightarrow e, t_2 \rightarrow s} st_3 \end{aligned} \tag{3}$$

Пусть $S_1 \dots S_N$ — подстановки переменных $v_1 \dots v_k$, $S_i = \{v_j \rightarrow P_{ij}\}$, где P — это некоторый образец с этими переменными. Обозначим за P_i подстановку S_i в образец P . Если $P \xrightarrow{S^*} P^*$, где $S^* = \{v_j \rightarrow P_j^*\}$, то $P^* \in \text{ЛСО}(P_1 \dots P_N)$ тогда и только тогда, когда $P_j^* \in \text{ЛСО}(P_{1j} \dots P_{Nj})$. Аналогично верно и для ГСО.

Определим *быстрое обобщение (БО)* для двух образцов P_1 и P_2 следующим образом:

- 1) если P_1 и P_2 являются термами, то БО определится согласно таблице 1.
- 2) Если P_1 и P_2 имеют следующий вид:

$$\begin{aligned} P_1 &= L_1^1 \dots L_{M_1}^1 e R_{N_1}^1 \dots R_1^1 \\ P_2 &= L_1^2 \dots L_{M_2}^2 e R_{N_2}^2 \dots R_1^2 \end{aligned}$$

то $\text{БО}(P_1, P_2) = L_1^* \dots L_{\min(M_1, M_2)}^* e R_{\min(N_1, N_2)}^* \dots R_1^*$, где $L_i^* = \text{БО}(L_i^1, L_i^2)$, $R_i^* = \text{БО}(R_i^1, R_i^2)$.

Таблица 1: Правила быстрого обобщения для двух термов

P_1	P_2			
	x	s	(P'_2)	t
x	x	s	t	t
$y \neq x$	s	s	t	t
s	s	s	t	t
(P'_1)	t	t	$(\text{БО}(P'_1, P'_2))$	t
t	t	t	t	t

3) Если P_1 и P_2 имеют вид:

$$P_1 = T_1^1 \dots T_k^1$$

$$P_2 = T_1^2 \dots T_k^2$$

то $\text{БО}(P_1, P_2) = T_1^* \dots T_k^*$, где $T_i^* = \text{БО}(T_i^1, T_i^2)$.

4) Во всех остальных случаях $\text{БО} = e$.

Заметим следующие свойства быстрого обобщения образцов:

— сложность алгоритма $O(\text{len}(P_1) + \text{len}(P_2))$, где $\text{len}(P)$ — длина образца образца P

— $\text{БО}(P_1, \text{БО}(P_2, P_3)) = \text{БО}(\text{БО}(P_1, P_2), P_3)$

Определим $\text{БО}(P_1 \dots P_N)$ как $\text{БО}(\text{БО}(P_1 \dots P_{N-1}), P_N)$.

$\text{БО}(P_1 \dots P_N) \Rightarrow^* \text{ЛСО}(P_1 \dots P_N)$

2.4 Сложнейший жесткий образец

Обозначим \mathbb{P} — множество всех образцов, \mathbb{H} — множество жестких образцов. *Сложнейший жесткий образец (СЖО)* $P_H \in \mathbb{H}$ для образца $P \in \mathbb{P}$ — это такой образец, что $P \Rightarrow^* P_H$ и $\nexists P'_H \in \mathbb{H} P \Rightarrow^* P'_H, C(P'_H) > C(P_H)$. На листинге 1 представлен псевдокод алгоритма получения сложнейшего жесткого образца.

Листинг 1 Алгоритм получения СЖО

```
1:   ▷  $S$  — множество подстановок
2:   ▷  $next$  — максимальный индекс переменной в  $P + 1$ 
3:   ▷  $P_H$  — сложнейший жесткий образец
4:  $P_H \leftarrow CHS(P)$ 
5: function  $CHS(P)$ 
6:   if  $P$  имеет вид  $(P')$  then
7:     return  $CHS(P')$ 
8:   if  $P$  имеет вид  $v_i$ , где  $v$  —  $s$  или  $t$  then
9:      $S \leftarrow \{v_{next} \rightarrow v_i\} \cup S$ 
10:     $next++$ 
11:    return  $v_{next}$ 
12:   if  $P$  имеет вид  $TP'$ , где  $T$  — терм then
13:     return  $CHS(T) + CHS(P')$ 
14:   if  $P$  имеет вид  $P'T$ , где  $T$  — терм then
15:     return  $CHS(P') + CHS(T)$ 
16:    $S = S \cup \{e_{next} \rightarrow P\}$ 
17:    $next++$ 
18:   return  $e_{next}$ 
19: end function
```

2.5 Классы образцов

Образец вида $L_1 \dots L_N e R_M \dots R_1$, где L_i, R_j — термы, будем называть *образцом класса* (M, N) .

Образец вида $T_1 \dots T_K$ будем называть *образцом класса* (K) .

Пусть $P_1 \dots P_K$ — образцы класса (N_i, M_i) , тогда ЛСО относится к классу $(\min(N_i), \min(M_i))$, $i = 1 \dots k$. Если же все образцы относятся к классу (K) , то аналогично предыдущему случаю их ЛСО относится к классу $(j, \min(K_i) - j)$, $i = 1 \dots k$, $j = 1 \dots \min(K_i)$.

Рассмотрим случаи, когда в исходном множестве образцов $P_1 \dots P_k$ встречаются образцы обоих классов. Пусть $K_{min} = \min(K_i)$, где индекс i пробегает по всем образцам класса (K) , а

M_{min} и N_{min} — соответственно минимум среди всех M и N образцов класса (M, N) . В таком случае возможны следующие варианты класса полученного ЛСО:

- $K_{min} > M_{min} + N_{min}$, ЛСО относится к классу (M_{min}, N_{min})
- $K_{min} < M_{min} + N_{min}$, и $K_{min} > M_{min}$, $K_{min} > N_{min}$, ЛСО относится к классам $(i, K_{min} - i)$, $i = K_{min} - N_{min} \dots M_{min}$
- $K_{min} < M_{min} + N_{min}$, и $K_{min} < M_{min}$, $K_{min} > N_{min}$, ЛСО относится к классам $(i, K_{min} - i)$, $i = K_{min} - N_{min} \dots K_{min}$
- $K_{min} < M_{min} + N_{min}$, и $K_{min} > M_{min}$, $K_{min} < N_{min}$, ЛСО относится к классам $(K_{min} - i, i)$, $i = K_{min} - M_{min} \dots K_{min}$
- $K_{min} < M_{min} + N_{min}$, и $K_{min} < M_{min}$, $K_{min} < N_{min}$, ЛСО относится к классам $(i, K_{min} - i)$, $i = 0 \dots K_{min}$

2.6 Оптимизация

На первом этапе оптимизации все образцы функции приводятся к жестким образцам. Для $i = 1 \dots N$ $P_{Hi} = \text{СЖО}(P_i)$, $P_{Hi} \xrightarrow{S_{Hi}} P_i$.

Следующий шаг заключается в вычислении быстрого обобщения $P^* = \text{БО}(P_{H1} \dots P_{Hn})$, $P^* \xrightarrow{S_i} P_{Hi}$, $S_i = \{v_j \rightarrow P_{ij} \mid j = 1 \dots k\}$ и генерации его кода. Для каждой v_j необходимо построить $P_j^* = \text{ГСО}(P_{1j} \dots P_{Nj})$. Рассмотрим процес вычисления глобального сложнейшего обобщения на следующем примере (4):

$$\begin{aligned} &\text{Первый образец: (А е.1) (В е.2)} \\ &\text{Второй образец: (В е.3)} \end{aligned} \tag{4}$$

Вычисляем быстрое обобщение и считаем классы, в которых нужно искать ЛСО.

Быстрое обобщение: $\text{е.Х} \rightarrow [(\text{А е.1) (В е.2)} \mid \text{В е.3}]$

Для образцов класса $(1, 1)$ и (1) ЛСО будут принадлежать либо классу $(1, 0)$, либо $(0, 1)$

Производим наложение на каждый из этих двух классов, после чего повторно вычисляем потермово ГСО и считаем сложность полученных образцов.

Класс $(1, 0)$

Наложения образцов будут иметь вид (А е.1) е.4 и (В е.3) е.5

Результат потермово вычисленного ГСО — (s.6 е.7) е.8

Сложность результата $C((\text{s.6 е.7) е.8}) = 1 + 0 + 2 + 0 + 3 - 2 = 4$

Класс $(0, 1)$

Наложения образцов будут иметь вид е.9 (В е.2) и е.10 (В е.3)

Результат потермово вычисленного ГСО — е.11 (В е.12)

Сложность результата $C(\text{е.11 (В е.12)}) = 1 + 0 + 0 + 3 + 3 - 2 = 5$

В итоге, из (s.6 е.7) е.8 и е.11 (В е.12) выбираем второй. ГСО((А е.1) (В е.2), (В е.3)) = е.11 (В е.12). По полученным БО и ГСО можно строить оптимизацию образцов функции.

3 Реализация алгоритма

3.1 Внутренне представление компилируемой программы

3.1.1 Лексический анализ

Перед оптимизацией очередной функции, происходит построение лексической свертки образцовых частей предложений. Каждая лексема является скобочным термом, снабженным тегом и атрибутом (в зависимости от типа лексемы (5)).

$$\begin{aligned} e.Tokens &::= t.Token^* \\ t.Token &::= (s.Tag\ e.Attr) \\ s.Tag &::= '('\ |'\ ')'\ |'\ \#S\ |\ \#T\ |\ \#E\ |\ \#Atom \end{aligned} \tag{5}$$

3.1.2 Синтаксический анализ

Синтаксический анализ строит синтаксическое дерево (6). Образец есть последовательность термов, терм — атом, переменная или скобки. Скобки — скобочный терм, внутри которого находится символ Brackets, и «тело» скобок, имеющего ту же структуру, что и для образца.

$$\begin{aligned} e.Pattern &::= t.Term^* \\ t.Term &::= t.Atom\ |\ t.Variable\ |\ (\#Brackets\ e.Pattern) \\ t.Atom &::= (\#Atom\ s.Attr) \\ t.Variable &::= (s.VariableType\ e.Index) \\ s.VariableType &::= \#S\ |\ \#E\ |\ \#T \end{aligned} \tag{6}$$

Для того, чтобы хранить подстановки, переводящие жёсткие образцы в реальные, формат переменной был изменен следующим образом:

$$t.Variable ::= (s.VariableType\ (e.Index)\ (e.Pattern))$$

3.1.3 Пример синтаксического дерева

На рисунке 2 показано синтаксическое дерево, построенное для образца:

$$((a))\ e.name1\ ((a)\ t.name2)$$

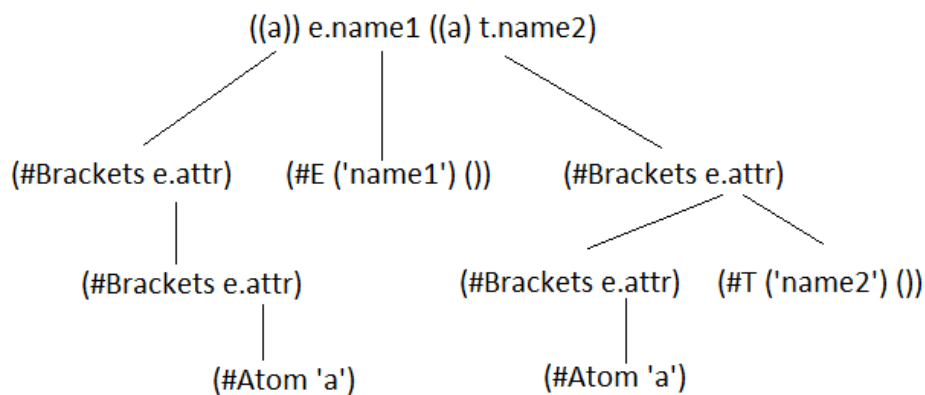


Рис. 2 – Пример синтаксического дерева

3.2 Обобщение образцов

3.2.1 Генерация жестких образцов

Как говорилось в пункте 2.6, на первом этапе оптимизации все образцы функции приводятся к жестким образцам. Алгоритм получения СЖО описан в пункте 2.4. Для того что бы не «протаскивать» между вызовами счётчик, инкрементируемый для каждой переменной, ее новое имя представляется уникальной последовательностью букв 'l', 'c', и 'r'. Функция генерации индексов принимает на вход образец, разбитый на лексемы и последним аргументом в структурных скобках — текущий индекс. Если образец можно разбить таким образом:

e.smth1 (e.smth2) e.smth3

то для каждой из частей происходит рекурсивный вызов функции генерации индексов со следующими добавлениями к текущему индексу

e.smth1 - текущий индекс + 'l' - спуск в левую часть

e.smth2 - текущий индекс + 'c' - спуск «по центру»

e.smth3 - текущий индекс + 'r' - спуск в правую часть

Если скобок нет, то отщипывается терм слева, а для оставшейся правой части опять происходит рекурсивный вызов, с добавлением буквы 'r' к текущему индексу. Если «оторванный» терм - переменная, то ее имя меняется на текущий индекс + 'l'.

В примере 7 показано, как будет выглядеть СЖО для образца с открытыми и повторными переменными.

$$\begin{aligned}
&\text{Образец: } (t.1) \text{ e.1 } () \text{ e.2 } () \text{ e.1} \\
&\text{Жесткий образец: } (t.\text{Index_l}) \text{ e.Index_r} \\
&\text{Подстановка для } t.\text{Index_l} \rightarrow [t.1] \\
&\text{Подстановка для } e.\text{Index_r} \rightarrow [e.1 \text{ } () \text{ e.2 } () \text{ e.1}]
\end{aligned}
\tag{7}$$

3.2.2 Быстрое обобщение

...

3.2.3 Глобальное сложнейшее обобщение

...

ЗАКЛЮЧЕНИЕ

Список литературы

- [1] РЕФАЛ [электронная публикация]. — URL: <http://refal.net/index.html>.
- [2] История рефал-компилятора [электронная публикация]. — URL:
<http://refal.net/romanenko/TheHistoryOfRefalCompiler/index.htm>.
- [3] Сопоставление с образцом [электронная публикация]. — URL: http://refal.net/rf5_frm.htm.