Федеральное государственное образовательное учреждение высшего профессионального образования

«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

Улучшенный алгоритм оптимизации совместного сопоставления с образцом в компиляторе Простого Рефала.

Руководитель курсового проекта _____ (А. В. Коновалов)
(подпись, дата)

Исполнитель курсового проекта,
студент группы ИУ9-72 _____ (П. А. Савельев)
(подпись, дата)

Москва, 2017

Содержание.

ВВЕДЕНИЕ.

Функциональное программирование — парадигма программирования, в которой вычисления выполняются над неизменяемыми переменными посредством вызова функций. К примеру, на данный времени широко известны такие языки функционального программирования как Lisp, Haskell, Scala.

РЕФАЛ (РЕкурсивных Функций АЛгоритмический) — один из старейших языков функционального программирования, который ориентирован на символьные вычисления. Позволяет выполнять обработку символьных строк, перевод с одного языка (искусственного или естественного) на другой. РЕФАЛ соединяет в себе практическую направленность на написание больших и сложных программ и математическую простоту. В 1970-1990 были реализованы несколько различных диалектов этого языка. В МГТУ имени Н. Э. Баумана на кафедре ИУ9 компилятор этого языка используется в качестве учебного полигона для курсовых и дипломных работ.

В связи с особенностями синтаксиса РЕФАЛ, предложения в одной и той же функции зачастую имеют схожую структуру. Зачастую такая ситуация возникает из-за того, что функции языка не могут принимать более одного аргумента. Вместо этого, из аргументов формируется единый объект, который разбивается на части уже непосредственно в теле функции.

Благодаря разработанным и реализованным ранее алгоритмам оптимизации обработки образцовых выражений удалось добавить механизм совместного сопоставления с образцом. Если все образцы функции имеют общий формат, то общий код сопоставления как бы "выносится за скобки" и выполняется лишь один раз. Однако, такая оптимизация не работает, к

примеру, в случаях, когда все предложения функции, кроме последнего, имеют схожую структуру, так как алгоритм пытается выделить общую часть у всех предложений. Описанная ситуация довольно часто встречается на практике.

Целью данной работы является разработка и реализация улучшенного алгоритма совместного сопоставления с образцом.

1 ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ.

1.1 Основные понятия языка РЕФАЛ.

В 1978 году С. А. Романенко опубликовал диссертацию «Машинно-независимый компилятор с языка рекурсивных функций». Как уже было отмечено ранее, в течение нескольких следующих лет были реализованы различные диалекты этого языка. Одним из них стал РЕФАЛ-5. В документации этого диалекта раскрываются базовые для многих других диалектов РЕФАЛ понятия. Рассмотрим их на примере РЕФАЛ-5:

- s - атом. Минимальная структурная единица.
- t - терм. Это s-переменная, атом, выражение в структурных или абстрактных скобках.
- e - выражение. Это последовательность атомов, термов или переменных. Последовательность может быть пустой.
- s, t и e переменные называют свободными. Свободные переменные состоят из указателя типа, точки и последовательности цифр и букв, называемой индексом переменной. Две переменные называются повторными, если их типы и индексы совпадают.
- Объектное выражение - это выражение, содержащее только структурные и абстрактные скобки, а также атомы.
- Образцом или образцовым выражением называется выражение, которое содержит помимо атомов, структурных и абстрактных скобок еще и свободные переменные.
- Результатом или результатным выражением называется выражение, которое содержит атомы, структурные, абстрактные скобки и скобки вызова функции.
- Конкретизация или вызов функции. Функция определяется при помощи двух скобок, заголовка и аргументов функции. В процессе работы РЕФАЛ-машины выполняется процедура конкретизации, которая

обозначается угловыми скобками. После открывающейся угловой скобки следуют заголовок функции, ее аргументы и закрывающаяся угловая скобка. В результате работы конкретизации, функция возвращает некоторое выражение, которое заменяет терм конкретизации в поле зрения. Если скобки конкретизации вложены в друг друга, РЕФАЛ-машина вызывает их поочерёдно, слева направо, начиная с самой внутренней.

- Важным понятием является процедура сопоставления с образцом. При запуске функции, выполняется попытка сопоставить входное выражение с левой частью первого предложения. Если сопоставление удаётся, то возвращается правая часть, иначе происходит переход к следующему предложению. Если ни одно предложение не может быть сопоставлено с аргументом функции, то программа завершает свою работу с ошибкой.
- Файл исходного текста на Рефале состоит из последовательности программных элементов — определений функций и ссылок на функции, определённые в других единицах трансляции.

Программа пишется в свободном формате, то есть переводы строк приравнены к обычным пробельным символам, там, где допустим пробельный символ (пробел или табуляция), допустима вставка перевода строки. Пробельные символы можно вставлять между двумя любыми лексемами языка. Пробелы обязательны в том случае, когда две лексемы, записываемые подряд, могут интерпретироваться как одна сплошная лексема (например, два числа, записанные слитно, будут интерпретироваться как одно число и т.д.). Пробелы недопустимы внутри идентификаторов, чисел, директив. Пробелы внутри цепочек литер интерпретируются как образы литер со значением «пробел».

1.2 Простой Рефал.

Простой Рефал - один из диалектов языка РЕФАЛ. К его отличительным особенностям можно отнести:

- Вложенные безымянные функции
- Идентификаторы не могут создаваться во время выполнения
- Отсутствие таких конструкций для функций как условия, блоки, действия

Если проводить сравнение Простого Рефала и РЕФАЛ-5, можно выделить следующие сходные черты:

- Использование терминов абстрактной РЕФАЛ-машины при определении семантики программы
- Сходный синтаксис функций и семантика сопоставления с образцом
- Использование объектных выражений в функциях

Среди отличительных особенностей Простого Рефала по отношению к РЕФАЛ-5 можно выделить:

- Абстрактные типы данных
- Вложенные функции
- Функции как атомы и как объекты первого класса

1.3 Структура компилятора Простого Рефала.

Компилятор Простого Рефала самоприменимый, производит генерацию кода на языке C++. В процессе многопроходной компиляции выполняются следующие шаги:

1. Загрузка исходного текста программы
2. Лексический анализ
3. Синтаксический анализ
4. Проверка контекстных зависимостей
5. Модуль удаления синтаксического сахара
6. Генерация высокоуровневого RASL
7. Генерация низкоуровневого RASL
8. Генерация целевого кода на C++

На этапе генерации высокоуровневого RASL по абстрактному

синтаксическому дереву строится промежуточное представление, которое представляет собой последовательность команд различного типа. На этом этапе происходит оптимизация совместного сопоставления с образцом.

Компилятор Простого Рефала поддерживает несколько режимов работы. Они определяются при помощи флагов указанных при компиляции. Существует два флага оптимизации: -OP и -OR, которые отвечают за оптимизацию образцовых и результатных выражений соответственно.

2 РАЗРАБОТКА АЛГОРИТМА.

2.1 Необходимые определения.

Для описания алгоритма улучшенной оптимизации совместного сопоставления с образцом требуется ввести следующие определения:

- Жёсткий образец — выражение без открытых и повторных переменных.
- Сложность образца — числовая характеристика образца, определяемая по формуле

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

где n_t , n_s , n_e – количество, соответственно, t-, s-, e- переменных, $n_{()}$ – количество пар скобок(как круглых, так и абстрактных), n_X — количество атомов.

- Подстановка — набор замен переменных в образце на некие образцы,

$$S=v_i \overset{S}{\to} P_i, i=1,\dots,N.$$

Запись $P_1 \overset{S}{\to} P_2$ означает, что P_2 получен из P_1 подстановкой S.

- Уточнение образца — операция замены переменных в образце. Образец P2 уточняет образец P1 , если существует подстановка S, переводящая P1 в P2.
- Обобщение образца — операция, обратная к уточнению.
- Пусть $P_1 \Rightarrow P_2$ - уточнение и не существует такого образца P_3, что $P_1 \Rightarrow P_3 \Rightarrow P_2$. Тогда $P_1 \Rightarrow P_2$ называется минимальным уточнением.
- Пусть $\mathcal{P}$ — множество всех жёстких образцов образца P. Сложнейшим жёстким образцом образца P называется образец $P` \in \mathcal{P}$, такой что не существует $Q \in \mathcal{P}$, что C(Q)>C (P`).

Теорема 1.

Минимальное уточнение возможно только следующим множеством замен:

- e → ε (пустота);

- e → te;
- e → et;
- t → (e);
- t → s;
- s → X (атом).

Пусть имеется набор образцов $P_1,...,P_N$.

- Локальное сложнейшее обобщение набора образцов $P_1,...,P_N$. (далее ЛСО) — такой образец P* , обобщающий каждый P_i из набора, что не существует образца Q, обобщающего каждый P_i из набора и P*

$$P^* \Rightarrow^* P_i, i=1,...,N \vee \nexists Q \Rightarrow^* P_i, i=1,...,N : Q \Rightarrow^* P^*.$$

- Глобальное сложнейшее обобщение набора образцов $P_1,...,P_N$. (далее ГСО) — такой образец P* , обобщающий каждый P_i из набора, что не существует образца Q, обобщающего каждый P_i из набора, сложность которого больше сложности P*:

$$P^* \Rightarrow^* P_i, i=1,...,N \vee \nexists Q \Rightarrow^* P_i, i=1,...,N : C(Q) > C(P^*).$$

Из определений следует, что ГСО∈ЛСО.

- Быстрое обобщение (далее БО) двух образцов P_1 и P_2 – образец P, построенный по определенным правилам:
  - если P_1 и P_2 являются термами, то быстрое обобщение строится согласно таблице 1;
  - если P_1 и P_2 являются образцами класса c(k), $P_1 = T_1^1 ... T_k^1$, $P_2 = T_1^2 ... T_k^2$, то $БО(P_1, P_2) = T_1^* ... T_k^*$, где $T_i^* - БО(T_i^1, T_i^2)$;
  - если P_1 и P_2 являются образцами класса $c(m_1,n_1)$ и $c(m_2,n_2)$ соответственно, $P_1 = L_1^1 ... L_{m_1}^1 e R_{n_1}^1 ... R_1^1$, $P_2 = L_1^2 ... L_{m_2}^2 e R_{n_2}^2 ... R_1^2$, то $БО(P_1, P_2) = L_1^* ... L_m^* e R_n^* ... R_1^*$, где $m=\min(m_1, m_2)$ $n=\min(n_1,n_2)$;
  - иначе, $БО(P_1, P_2)=e$.

Быстрое обобщение набора образцов $P_1,...,P_N$ определяется рекурсивно:

- $БО(P_1,...,P_{N-1}, P_N)=БО(БО(P_1,...,P_{N-1}), P_N)$.

Таблица 1. Построение быстрого обобщения двух термов.

P_1/ P_2	X	Y!=X	s	t	(P_1)
X	X	s	s	t	t
s	s	s	s	t	t
t	t	t	t	t	t
(P_2)	t	t	t	t	$(БО(P_1, P_2))$

Теорема 2.

$БО(P_1,...,P_N) \Rightarrow$ * $ГСО(P_1,...,P_N)$.

2.2 Текущая реализация оптимизации совместного сопоставления с образцом.

Генерация жёстких образцов для списка образцовых частей предложений функции производится в модуле HardSent.sref. В качестве входных данных заглавная функция HardSent модуля получает образец, жёсткий образец которого необходимо построить. В качестве результата функция возвращает жёсткий образец и подстановки переменных этого образца, переводящие его в исходный образец.

При составлении жёсткого образца получается образец вида:

$$P_H = \left(v_1\left('idx'\right)\left(\left(P_1'\right)\right)\right)...\left(v_n\left('idx'\right)\left(\left(P_n'\right)\right)\right),$$

где v_i – тэг типа переменной, P_i' – подстановка переменной v_i в образец P_H.

Далее в качестве входных данных основная функция FastGen модуля FastGen.sref получает список жёстких образцов и замен переменных для них. Этот модуль выполняет вычислении быстрого обобщения списка образцовых выражений. Как и многие другие функции, FastGen является рекурсивной. Функция находит быстрое двух первых элементов списка, после чего помещает результат в начало этого же списка вместо обработанных образцов и вызывается рекурсивно.

Следующим этапом происходит вычисление глобального сложнейшего

обобщения в модуле GlobalGen.sref. Для каждой замены проводится вычисление класса, для всех замен — вычисление допустимых классов частичного ГСО, после ищутся потермовые обобщения, сложнейшее из которых и возвращается.

В генерации кода участвует функция GenPatern, которая вызывается не для каждого предложения функции, а отдельно для общих частей и различающихся частей предложений. При вызове для общих частей предложений в качестве аргумента передаётся составленное глобальное сложнейшее обобщение. На его основе формируется список команд сопоставления выражения с обобщением. После составления промежуточного представления для общей части предложений, GenPatern вызывается для генерации оставшихся команд для каждого предложения. При этом она получает в качестве аргумента список подстановок и генерирует на их основе команда сохранения и сопоставления. В конечном итоге команды сопоставления одинаковых элементов будут вынесены «за скобки».

2.3 Описание алгоритма.

Так как для улучшения алгоритма совместного сопоставления с образцом требуется выделять группы предложений с похожей структурой, анализ разбиения групп будет производиться на основании подстановок, сформированных после вычисления глобального сложнейшего обобщения.

Каждую подстановку требуется обобщить до e- или t-переменной. Это позволит исключить ситуации, когда из-за недостаточного обобщения предложения, обладающие похожей структурой, не были бы объединены в одну группу.

Следующим этапом становится вычисление количества предложений в первой группе благодаря использованию рекурсии по предложениям, содержащим подстановки. В случае недостаточного различия между подстановками, функция должна возвращать общее количество предложений. Тогда полученная единственная группа будет обрабатываться так же, как

обрабатывалась ранее до улучшения алгоритма.

После того, как будет произведено разбиение исходных предложений на две группы, для каждой группы нужно вычислить глобальное сложнейшее обобщение.

Сопоставляя обобщение для каждой группы и обобщение для всех предложений, получим новые подстановки, обладающие одинаковой левой частью. Правая часть этих подстановок будет содержать переменные, сгенерированные для каждой группы независимо друг от друга. После этого нужно переименовать переменные, чтобы исключить совпадение их названий в дальнейшем при генерации кода.

На этапе генерации кода нужно выполнять композицию команд, сгенерированных для обобщения всех предложений с командами, сгенерированными для обобщения предложений каждой из групп. Требуется передавать вершину контекста, для вычисления корректного смещения переменных.

3 РЕАЛИЗАЦИЯ АЛГОРИТМА.

3.1 План внедрения.

Добавим новую функцию HighLevelRASL-Function-Conjoint-Extra, которая в начальном приближении будет вызывать исходную функцию HighLevelRASL-Function-Conjoint (см. Листинг 1).

Листинг 1. Функция HighLevelRASL-Function-Conjoint-Extra

```
1.$ENTRY HighLevelRASL-Function-Conjoint-Extra {
2.  s.FnGenInitSubst s.FnGenSubst s.FnGenResult
3.  s.ScopeClass (e.Name) e.Sentences
4.    = <HighLevelRASL-Function-Conjoint
5.        s.FnGenInitSubst s.FnGenSubst
6.        s.FnGenResult
7.        s.ScopeClass (e.Name) e.Sentences
8.     >;
9.}
```

Определим новый флаг "OE" в списке флагов оптимизации. Этот флаг будет соответствовать оптимизации образцовых выражений при помощи определённой в Листинге 1 функции.

В процессе выполнения функции HighLevelRASL-Function-Conjoint происходит генерация обобщения образцовых выражений для всех предложений, после чего для каждого предложения генерируются подстановки, уточняющие переменные из обобщения. Для того, чтобы разбить исходные предложения на 2 группы, требуется предварительно обобщить подстановки для каждого образца.

Следующим этапом становится определение количества предложений в первой группе и разбиение предложений на две группы. Выполнив глобальное сложнейшее обобщение для каждой группы по отдельности, мы получим обобщение образцовых выражений для этих групп.

Анализируем общие части образцовых выражений и дополняем сформированные группы новой подстановкой.

3.2 Добавленные функции.

Обобщение подстановки происходит в несколько этапов:

1. Выделение подстановки из каждого предложения.
2. Производим изменение формата подстановки.
3. Обобщение до последовательности термов и e-переменных.
4. Обобщение комбинаций 'et', 'te' и 'ee'.

Функция на Листинге 2 вызывается для каждой подстановки, заменяя ее на обобщение.

Листинг 2. Функции для обобщения подстановки

```
1.CustomPatternComment-Main {
2.  e.Pattern
3.    = <CustomPatternComment-Complex
4.        <CustomPatternComment-Simple
5.          e.Pattern
6.        >
7.     >;
8.}
9.
10.CustomPatternComment-Simple {
11.  e.Pattern
12.    = <Map
13.        {
14.          (#Atom  e.all) = (#T);
15.          (#S '$' e.all) = (#T);
16.          (#T '$' e.all) = (#T);
17.          (/* empty  */) = (#E);
18.          (#E '$' e.all) = (#E);
19.          (#Brackets e.all) = (#T);
20.          (#ADT-Brackets e.all) = (#T);
21.        }
22.        e.Pattern
23.      >;
24.}
25.
26.CustomPatternComment-Complex {
27.  e.Pattern
28.    = <Fetch e.Pattern
29.        {
30.          (#T) (#E) e.other
```

```
31.              = <CustomPatternComment-Complex
32.                  (#E) e.other
33.                >;
34.            (#E) (#T) e.other
35.              = <CustomPatternComment-Complex
36.                  (#E) e.other
37.                >;
38.            (#E) (#E) e.other
39.              = <CustomPatternComment-Complex
40.                  (#E) e.other
41.                >;
42.            (e.1) e.else
43.              = e.1
44.                <CustomPatternComment-Complex
45.                  e.else
46.                >;
47.            /*термы кончились*/ = ;
48.          }
49.       >;
50. }
```

При несовпадении обобщённых подстановок функция GetSplitNum (см. Листинг 3) вернёт номер последнего предложения первой группы. Если все предложения имеют совпали, то функция вернёт их в качестве первой группы.

Листинг 3. Определение номера разбиения групп предложений

```
1.GetSplitNum {
2.   ((e.common) s.num #Continue) e.Tails () =
3.     s.num;
4.
5.   ((e.common) s.num #Continue) e.Tails
6.     = <GetSplitNum
7.         <GetSplitNum-CompareCommon e.Tails>
8.       >;
9.
10.   ((e.common) s.num #Break) e.Tails = s.num;
11.
12.   e.Sentences = <GetSplitNum
```

```
                    <GetSplitNum-CompareCommon
                      e.Sentences
                    >
                  >;
}

GetSplitNum-CompareCommon {
  e.Sentences
    = <MapReduce
        {
          (() 0 #Continue)(t.common e.tail)
            = ((t.common) 0 #Continue)(e.tail);

          ((t.common) s.num #Continue)
          (t.common e.tail)
            = (
                (t.common)
                <Inc s.num> #Continue
              )(e.tail);

          ((t.common) s.num #Continue)(e.tail)
            = ((t.common) s.num #Break)();

          ((t.common) s.num #Break)(e.tail)
            = ((t.common) s.num #Break)();
        }
        (() 0 #Continue)
        e.Sentences
      >;
}
```

Разбиение предложений на две группы по номеру выполняется функцией Split-Sentences (см. Листинг 4):

Листинг 4. Разбиение предложений на две группы по номеру

```
Split-Sentences {
  e.Sentences s.Num
    = <Split-DivideGroups
        <Split-RemainingSentences
          e.Sentences s.Num
```

```
6.          >
7.       >;
8.}
9.
10. Split-RemainingSentences {
11.   t.Sentence e.Sentences 0
12.     = t.Sentence (e.Sentences);
13.
14.   t.Sentence e.Sentences s.Num
15.     = t.Sentence
16.       <Split-RemainingSentences
17.         e.Sentences <Dec s.Num>
18.       >;
19. }
20.
21. Split-DivideGroups {
22.   e.FirstSentences (e.SecondSentences)
23.     = (e.FirstSentences)(e.SecondSentences);
24. }
```

После разбиения групп и нахождения глобального сложнейшего обобщения для каждой группы по отдельности, ищется обобщающая подстановка для каждой группы (см. Листинг 5). Такая подстановка формируется из первоначального обобщения для всех предложений и обобщения предложений для группы.

Листинг 5. Формирование обобщающей подстановки

```
1.GetNewSubst {
2.   (t.Pattern e.CommonTail)
3.   (t.Pattern e.CustomTail)
4.     = <GetNewSubst
5.         (e.CommonTail) (e.CustomTail)
6.       >;
7.   ((e.PatternCommon) )
8.   (e.PatternCustom e.CustomTail)
9.    = (
10.         e.PatternCommon '$' e.PatternCustom
11.         e.CustomTail
12.       );
13. }
```

ЗАКЛЮЧЕНИЕ.

В данной работе была реализована часть алгоритма, отвечающая за обобщение подстановок, сформированных после вычисления глобального сложнейшего обобщения; реализовано разбиение исходных предложений на группы для последующего обобщения, а также дополнение подстановок подстановкой, содержащей общую левую часть для каждой из групп.

При этом в дальнейшем планируется изменить генерацию кода для того, чтобы объединять списки команд сопоставления выражения с обобщением для каждой группы с общим списком команд; добавить рекурсивное разбиение групп предложений, поработать над производительностью, чистотой кода.

Список использованной литературы.

1. Турчин В. Ф. Пользовательская документация для языка РЕФАЛ-5 [Электронный ресурс]: Содружество «РЕФАЛ/Суперкомпиляция» .- Режим доступа: http://www.refal.net/rf5_frm.htm .
2. Турчин В. Ф. Алгоритмический язык рекурсивных функций (РЕФАЛ). — М.: ИПМ АН СССР, 1968.
3. А. В. Коновалов, Компилятор Простого Рефала. [Электронный ресурс]. URL: https://github.com/Mazdaywik/simple-refal (дата обращения: 24.03.2017).
4. Коновалов А.В. Пользовательская документация для языка Простой Рефал [Электронный ресурс]: GitHub .- Режим доступа: https://github.com/bmstu-iu9/simple-refal.
5. Романенко С. А. Машинно-независимый компилятор с языка рекурсивных функций, 1978.