Выпускная квалификационная работа по теме "Оптимизация сопоставления с образцом в компиляторе языка Рефал".

Студент: Скрыпников И. Е.
Руководитель: Коновалов А. В.

Москва, 2016

Цель работы

Разработать и реализовать алгоритм оптимимизации сопоставления с образцом в компиляторе языка Простой Рефал.

Постановка задачи

Образец включает несколько переменных.
Сопоставление — поиск их значений.

```
Func {
    Pattern1 = Result1;
    Pattern2 = Result2;
...
    PatternN = ResultN;
}

<Func Argument>;
```

Необходимые определения и понятия

Определение 1. Жесткий образец – образец, переменные которого определяются однозначно и не повторяются.

Определение 2. Подстановка – набор значений переменных v_i в образце P_1 $S=\{v_i \to P_i | i=1,\dots,N\}$, переводящих образец P_1 в образец P_2, $P_1 \overset{S}{\to} P_2$.

Определение 3. Обобщением образца P_2 называется такой образец P_1, что $P_1 \overset{S}{\to} P_2$.

Обобщение обозначается $P_1 \Rightarrow^* P_2$.

Необходимые определения и понятия

Определение 4. Сложность образца – числовая характеристика, определяемая по формуле $C(P)=n_t+2n_s+3n_X+3n_{()}-n_e+1$, где n_t, n_s, n_e – кол-во, соответственно, t-, s-, e-переменных, n_0 – кол-во пар скобок, n_X — кол-во атомов в образце P.

Определение 5. Глобальное сложнейшее обобщение (ГСО) набора образцов P_1, ..., P_N — образец P такой, что

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

Необходимые определения и понятия

Определение 6. Быстрое обобщение (далее БО) двух образцов P_1 и P_2 – образец P, построенный по определенным правилам:

1) если P_1 и P_2 являются термами, то быстрое обобщение строится так:
   - Если P_1=(P_1') и P_2=(P_2'), то БО(P_1, P_2)=(БО(P_1', P_2'));
   - Если один из образцов является s-переменной, второй – атом или s-переменная, то БО(P_1, P_2) – s-переменная. Для пары разных атомов БО – s-переменная, а одинаковых – атом.
2) Если $P_1 = T_1^1 \dots T_k^1, P_2 = T_1^2 \dots T_k^2$, где T_i^j – термы, то $БО(P_1, P_2) = T_1^* \dots T_k^*, T_i^* = БО(T_i^1, T_i^2)$;
3) Если $P_1 = L_1^1 \dots L_m^1 e R_n^1 \dots R_1^1, P_2 = L_1^2 \dots L_m^2 e R_n^2 \dots R_1^2$, где L_i^j, R_i^j – термы, $БО(P_1, P_2) = L_1^* \dots L_m^* e R_n^* \dots R_1^*, L_1^* = БО(L_i^1, L_i^2), R_1^* = БО(R_i^1, R_i^2)$;
4) Иначе, $БО(P_1, P_2) = e$.

Быстрое обобщение набора образцов P_1, …, P_N определяется рекурсивно: $БО(P_1, \dots, P_{N-1}, P_N) = БО(БО(P_1, \dots, P_{N-1}), P_N)$.

Используемые теоремы

Теорема 1 (об обобщении подстановок).

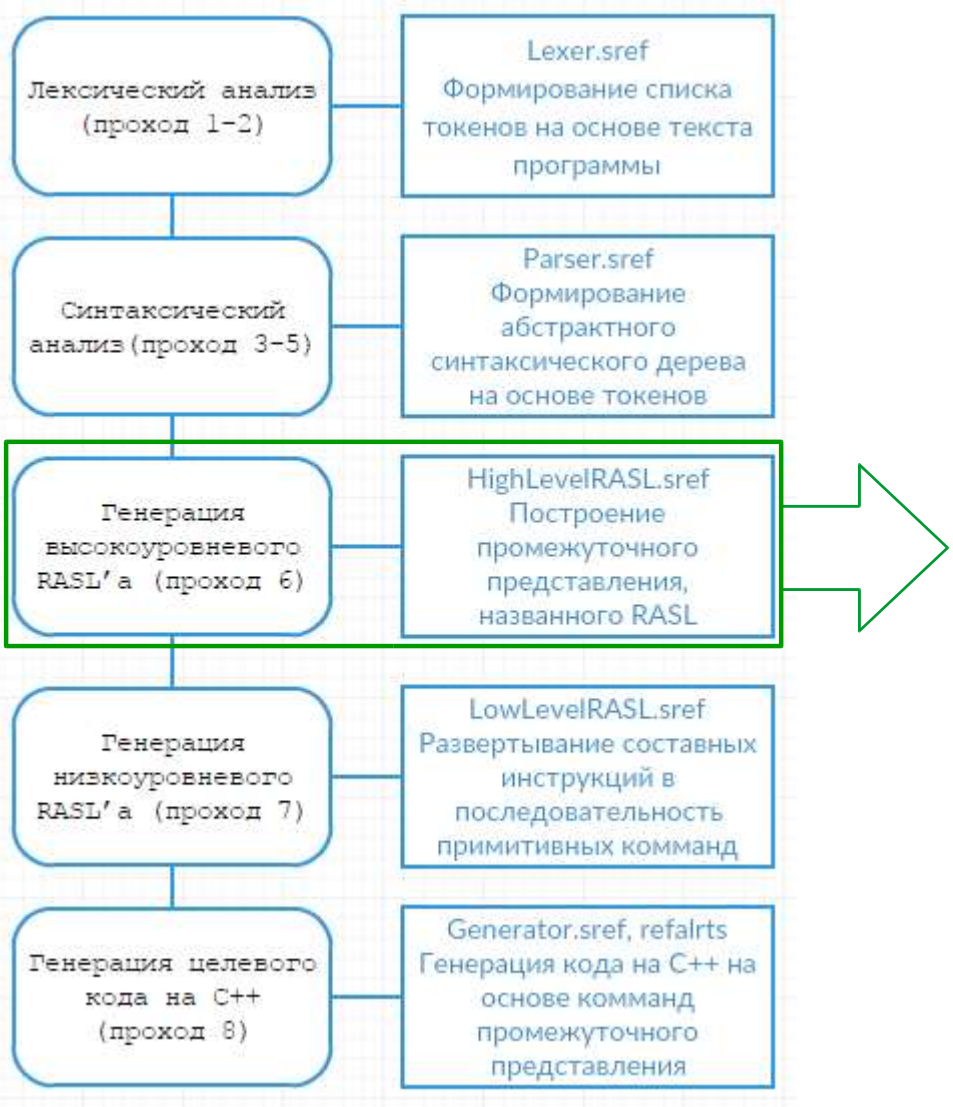
Пусть P – образец, содержащий переменную v, существуют подстановки $S^* = \{ v \to P^{*\prime} \}, P \overset{S^*}{\to} P^*$ и $S_1, \ldots, S_N, S_i = \{ v \to P_i' \}, P \overset{S_i}{\to} P_i$. В этом случае,

$$P^* \in ГСО(P_1, \ldots, P_N) \Leftrightarrow P^{*\prime} \in ГСО(P_1', \ldots, P_N').$$

Теорема 2 (о связи быстрого и глобального обобщений).

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

Схема работы компилятора



В модуле `HighLevelRasl.sref` компилятора Простого Рефала проводится преобразование абстрактного дерева в промежуточное представление. Получая на вход каждое предложение функции в формате `((e.Pattern)(e.Result))`, функция `HighLevelRasl-Function` возвращает команды RASLa, отвечающие за сопоставление аргумента с переменными образца.

После оптимизации предложения передаются в другом формате: `(e.Patterns)(e.Results)`. Теперь они обрабатывается одновременно, что позволяет сопоставлять одинаковые или похожие по структуре образцы вместе.

Команды промежуточного представления являются основой для дальнейшей генерации кода на языке C++.

Изменения в сгенерированном коде

До оптимизации

```
do {
    if( !char_left( 'a' ))
        continue;

    if( !char_left( 'b' ))
        continue;

} while ( 0 );

do {
    if( !char_left( 'a' ))
        continue;

    if( !char_left( 'd' ))
        continue;

} while ( 0 );
```

После оптимизации

```
if( !char_left( 'a' ))
  return cRecognitionImpossible;

do {
    if( !char_left( 'b' ))
        continue;

} while ( 0 );

do {
    if( !char_left( 'd' ))
        continue;

} while ( 0 );
```

Результаты тестирования

	Без оптимизации	С оптимизацией	Δ
BracketCounter	4,081c	3,802c	0,279c
LetterCounter	43,412c	35,582c	7,830c
Lexer	6,366c	4,207c	2,159c

Заключение

В работе был разработан и реализован алгоритм оптимизации сопоставления аргумента с образцом в компиляторе языка Простой Рефал. Результаты тестирования показывают, что алгоритм применим и эффективен. Кроме того, описанный теоретический комплекс может быть использован при создании других оптимизаторов и верификаторов программ на языке Простой Рефал.