Московский Государственных Технический Университет имени Н.Э. Баумана
Факультет: «Информатика и системы управления»
Кафедра: «Теоретическая информатика и компьютерные технологии»

Алгоритм прогонки вызовов рекурсивных функций, не приводящий к зацикливанию в компиляторе Рефала-5λ

Выполнил:
Студент группы ИУ9-82Б
Апахов М. Д.

Научный руководитель:
Старший преподаватель
Коновалов А. В.

Москва, 2021

Постановка задачи

1. Модификация алгоритма оптимизации прогонки и встраивания вызовов функций в компиляторе языка Рефал-5λ.
2. Оптимизация самоприменимого компилятора языка Рефал-5λ с помощью разработанных решений.
3. Оценка результатов оптимизации прогонки и встраивания функций.

Язык Рефал-5λ

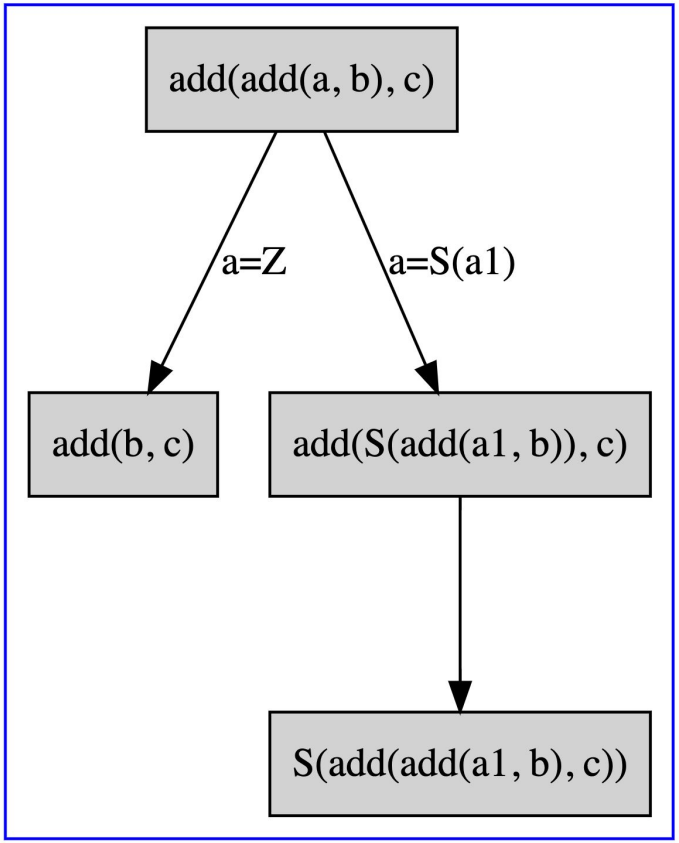
1. Функциональный.
2. Компилируемый.
3. Пример функции

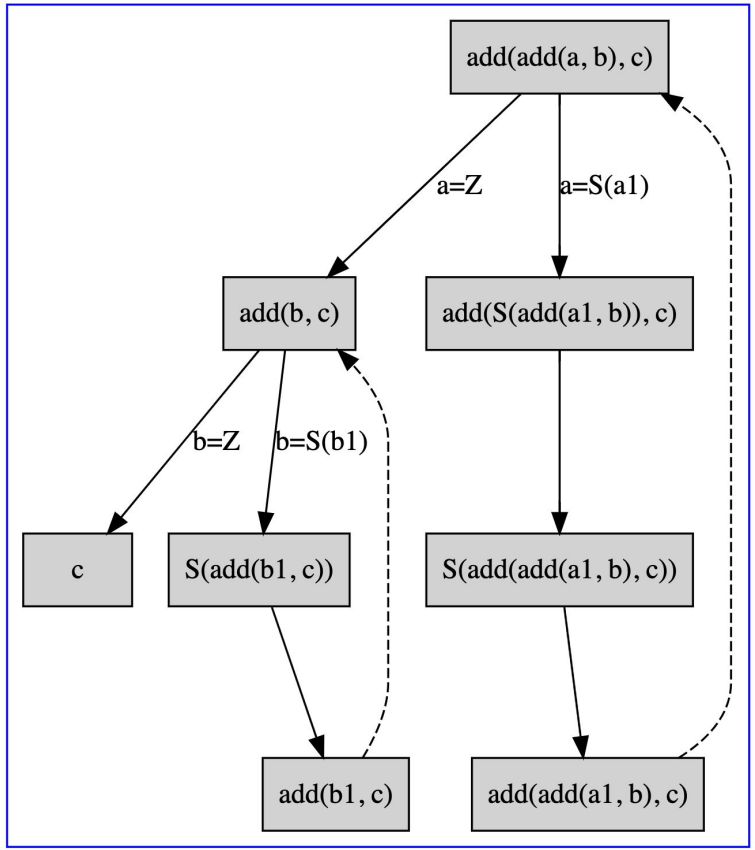
```
IsPalindrome {
    s.1 e.m s.1 = <IsPalindrome e.m>;
    s.1 = True;
    /* empty */ = True;
    e.Other = False;
}
```

Суперкомпиляция

1. Что значит термин?
2. Для чего используется?
3. Краткая суть

Граф конфигураций





Отношение Хигмана-Крускала

1. $x \trianglelefteq y$ для любых переменных x и y .
2. $X \trianglelefteq f(Y1, \dots, Yn)$, если f имя конструктора или функции и существует i такое, что $X \trianglelefteq Yi$.
3. $f(X1, \dots, Xn) \trianglelefteq f(Y1, \dots, Yn)$, если f имя конструктора или функции и для любого i n верно, что $Xi \trianglelefteq Yi$.

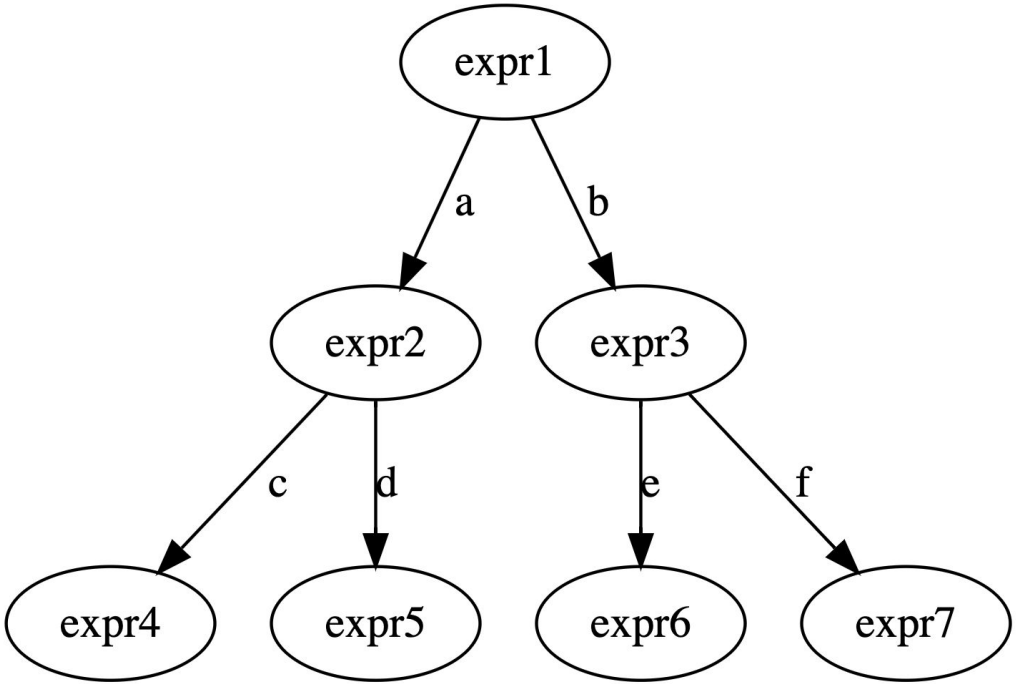
Старый алгоритм прогонки

1. Рассматриваем предложение L = R
2. Получаем набор решений.
3. Одним решением является выражение и набор сужений {E, C={C1,...,CN}}
4. Для каждого решения получаем новое предложение L*C = E

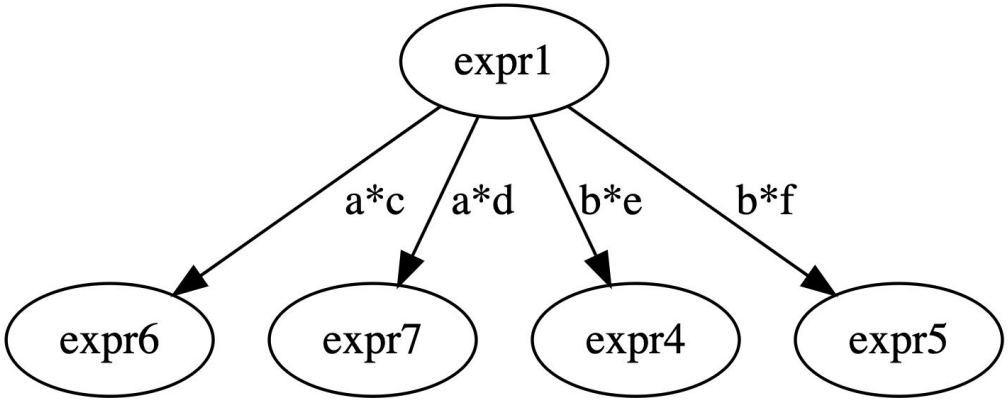
Новый алгоритм прогонки

1. Рассматриваем предложение L = R .
2. Получаем набор решений.
3. Одним решением является выражение и набор сужений {E, C={C1,...,CN}} .
4. Для каждого выражения в решениях продолжаем прогонку. Получаем граф конфигураций для выражения R.

Новый алгоритм прогонки. Граф конфигураций



Новый алгоритм прогонки. Граф конфигураций



Реализация нового алгоритма

1. Функция MakeDriveTree
2. Функция FlatDriveTree
3. Функция ColdTree

Композиция сужений

$$V_1^1 : R_1^1$$
$$V_2^1 : R_2^1$$
$$\dots$$
$$V_n^1 : R_n^1$$

Набор C1

$$V_1^2 : R_1^2$$
$$V_2^2 : R_2^2$$
$$\dots$$
$$V_m^2 : R_m^2$$

Набор C2

$$V_1^1 : R_1^1 * (V_1^2 : R_1^2) * (V_2^2 : R_2^2) * \dots * (V_m^2 : R_m^2)$$
$$V_2^1 : R_2^1 * (V_1^2 : R_1^2) * (V_2^2 : R_2^2) * \dots * (V_m^2 : R_m^2)$$
$$\dots$$
$$V_n^1 : R_n^1 * (V_1^2 : R_1^2) * (V_2^2 : R_2^2) * \dots * (V_m^2 : R_m^2)$$
$$V_1^2 : R_1^2$$
$$V_2^2 : R_2^2$$
$$\dots$$
$$V_m^2 : R_m^2$$

Набор композиции C1*C2

Тестирование. Пример 1. Код

```
$DRIVE INC;
INC {
    (t.X) = <Add t.X 1>;
}

$ENTRY Go {
    = <Prout <MAP &INC (1) (2) (3) /* ещё множество аналогичных выражений */ >>
  ;
}

$DRIVE MAP;
MAP {
    t.Closure t.Elem e.Rest =
        <t.Closure t.Elem> <MAP t.Closure e.Rest>
    ;
    t.Closure =
    ;
}
```

Тестирование. Пример 1. Вывод

```
$ENTRY Go {
  /* empty */ = <Prout <MAP &INC (1) (2) (3)>>;
}
.........
$ENTRY Go {
  /* empty */ = <Prout 2 3 4>;
}
```

Тестирование. Пример 2. Код

```
$DRIVE ADD;
ADD {
    (Z) (e.X) = e.X
    ;
    ('1' e.N) (e.X) = '1' <ADD (e.N) (e.X)>
    ;
}

$ENTRY Go {
  = <Prout <ADD ('11' Z) ('111' Z)>>
  ;
}
```

Тестирование. Пример 2. Вывод

```
$ENTRY Go {
  /* empty */ = <Prout <ADD ('11' Z) ('111' Z)>>;
}
........
$ENTRY Go {
  /* empty */ = <Prout '11111' Z>;
}
```

Заключение

1. Компилятор языка Рефал-5λ был модифицирован: реализован, отлажен и протестирован новый алгоритм прогонки, который не приводит к зацикливанию на любой функции в языке Рефал-5λ.
2. Алгоритм оптимизации прогонки был применен к более широкому спектру функций в компиляторе языка. Также алгоритм позже позволит уменьшить количество итераций на этапе древесных оптимизаций в компиляторе, что может дать прирост в скорости компиляции программ на языке Рефал-5λ.

Спасибо за внимание