

Прогонка и встраивание функций в языке РЕФАЛ-5λ

Студент группы ИУ9-82: Ситников К.А.

Научный руководитель: Коновалов А.В

Постановка задачи

- Разработка алгоритма оптимизации встраивания и прогонки функций в компиляторе.
- Изменение грамматики языка РЕФАЛ-5λ и организация поддержки новых синтаксических конструкций, необходимых для реализации оптимизации в компиляторе.
- Оптимизация самоприменимого компилятора языка РЕФАЛ-5λ с помощью разработанных решений.
- Оценка результатов оптимизации прогонки и встраивания.

Обзор языка

РЕФАЛ-5λ – язык, основанный на сопоставлении с образцом.

Основные конструкции языка – функции.

Функция состоит из набора предложений, где левая часть – выражение-образец, а правая представляет комбинацию частей, полученных при сопоставлении с образцом.

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

Встраивание

Встраивание – замена вызова функции на правую часть её предложения, с левой частью которого успешно сопоставляется аргумент вызова

```
J {  
    s.2 s.3 = s.3 <N s.2 s.2>  
}  
  
N {  
    s.u s.u = s.u True; e.Other = ;  
}  
  
/* Преобразование */  
  
J {  
    s.2 s.3 = s.3 s.2 True  
}
```

Прогонка

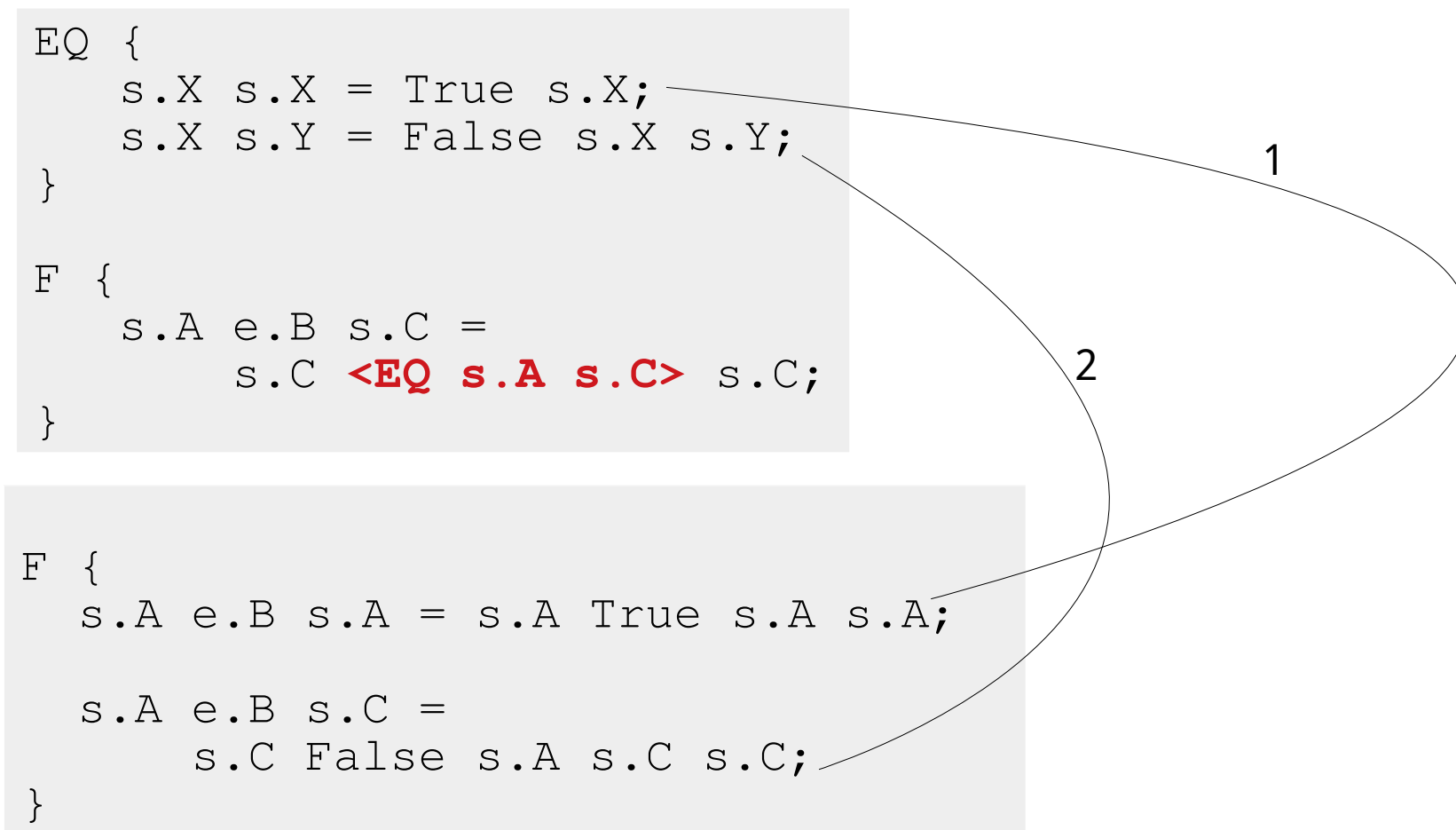
Прогонка – преобразование функции, в которой вызовы других функций в правой части предложений устраняются, при этом семантические свойства функции не изменяются.

```
EQ {  
  s.X s.X = True s.X;  
  s.X s.Y = False s.X s.Y;  
}  
  
F {  
  s.A e.B s.C =  
    s.C <EQ s.A s.C> s.C;  
}
```

```
F {  
  s.A e.B s.A = s.A True s.A s.A;  
  
  s.A e.B s.C =  
    s.C False s.A s.C s.C;  
}
```

1

2



Алгоритм обобщенного сопоставления

```
F {  
  e.Expr = e.LeftF <G e.Args> e.RightF;  
  ...  
}  
  
G {  
  e.Left1 = e.Right2;  
  ...  
}
```

1. Алгоритм на основе решения **e.Args** : **e.Left1** предоставляет информацию о том, как функция должна быть преобразована.
2. Решение представляет собой набор сужений и присваиваний.
3. Сужения – информация о преобразовании левой и правой части до и после вызова. Вид: $[V \rightarrow \text{Expr}]$, **v** – переменная, **Expr** – некоторое выражение.
4. Присваивания – информация о преобразовании вызова. Вид: $(\text{Expr} \leftarrow v)$

Эквивалентные преобразования

```
F {  
  e.Expr = e.LeftF <G e.Args> e.RightF;  
  ...  
}  
  
G {  
  e.Left1 = e.Right2;  
  ...  
}
```

Пусть A – набор присваиваний, C – набор сужений.

```
F' {  
  C // e.Expr = C // e.LeftF A // e.Right2 C // e.RightF;  
  ...  
}
```



Оператор // - применение присваиваний и сужений вида $v \rightarrow Expr$.

Все вхождения переменной v заменяются на $Expr$

F и **F'** - эквивалентны. То есть, для любых входных данных будут давать одинаковый результат.

Алгоритм обобщенного сопоставления

```
EQ {  
  s.X s.X = True s.X;  
  s.X s.Y = False s.X s.Y;  
}  
  
F {  
  s.A e.B s.C =  
    s.C <EQ s.A s.C> s.C;  
}
```

Необходимо решить
следующие уравнения:

1. **s.A s.C** : **s.X s.X**

2. **s.A s.C** : **s.X s.Y**

РЕШЕНИЕ:

1. $[s.C \rightarrow s.A], (s.A \leftarrow s.X)$
2. $(s.A \leftarrow s.X), (s.C \leftarrow s.Y)$

```
F {  
  s.A e.B s.A = s.A True s.A s.A;  
  
  s.A e.B s.C =  
    s.C False s.A s.C s.C;  
}
```


Расширения синтаксиса

```
$DRIVE G;
```

```
G {  
    . . .  
}
```

```
$INLINE F;
```

```
F {  
    . . .  
}
```

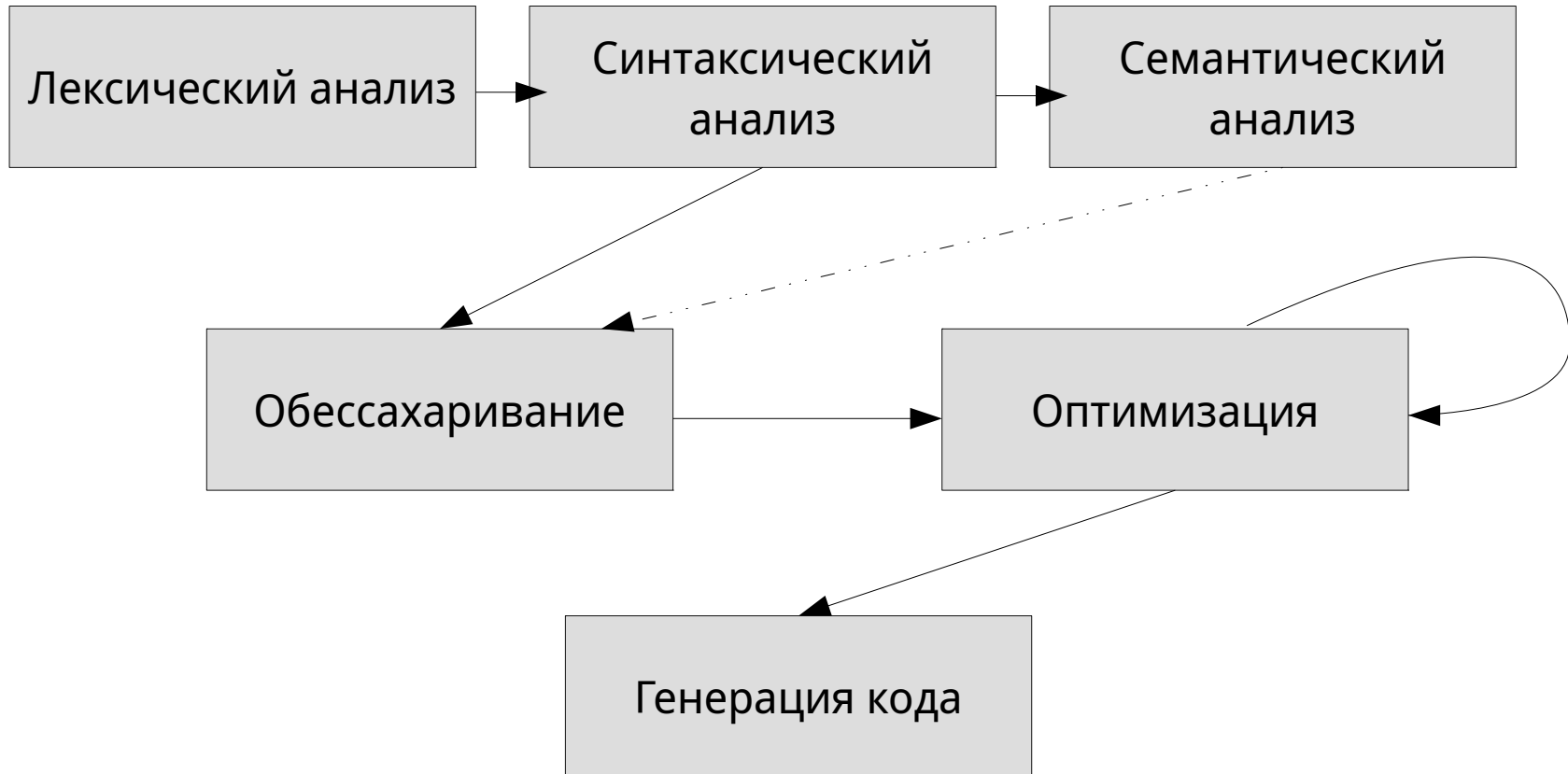
В рамках работы были разработаны расширения синтаксиса для применения оптимизаций – ключевые слова `$DRIVE` и `$INLINE`.

Если у функции есть метка `$DRIVE`, то ее вызовы прогоняются.

Если у функции есть метка `$INLINE`, то ее вызовы встраиваются.

Оптимизации можно включить, передав специальные флаги компилятору.

Компилятор языка РЕФАЛ5-λ

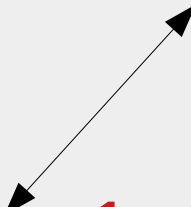


Стадия **оптимизации** выполняется несколько раз до тех пор, пока синтаксическое дерево не перестанет меняться, или компилятор не достигнет лимита итераций оптимизации.

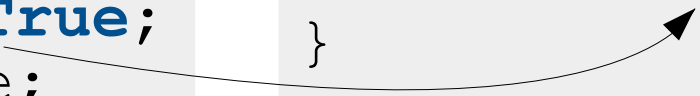
Прогонка

Результат решения однозначен: Success (() (A))

```
F {  
  s.A = <G s.A s.A>  
}  
  
G {  
  s.1 s.1 = s.1 True;  
  s.1 s.2 = False;  
}  
  
$DRIVE G;
```



```
F {  
  s.A = s.A True;  
}
```

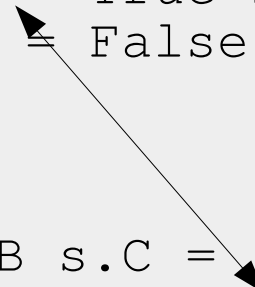


Прогонка

Результат решения неоднозначен:

Success ((C_1) (A_1)) .. ((C_K) (A_K))

```
EQ {  
  s.X s.X = True s.X;  
  s.X s.Y = False s.X s.Y;  
}  
  
F {  
  s.A e.B s.C =  
    s.C <EQ s.A s.C> s.C;  
}
```



/* После преобразования */

```
F {  
  s.A e.B s.A = s.A True s.A s.A;  
  s.A e.B s.C =  
    s.A <EQ*1 s.A s.C> s.A;  
}
```

```
EQ*1 { s.X s.Y = False s.X s.Y; }
```

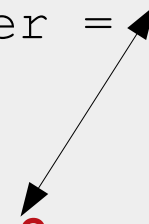
В этом случае после предложения, подвергнутого оптимизации, добавляется предложение с вызовом функции **EQ*1**, которая является функцией **EQ** без первого предложения.

Таким образом, обеспечивается решение уравнений для других предложений функции **EQ** на следующих итерациях оптимизации

Прогонка

Результат решения уравнения - Failure

```
F {  
  s.A = <G A s.A B>  
  e.Other = e.Other;  
}  
  
G {  
  s.1 e.2 s.1 = True;  
}  
  
/* После преобразования */  
  
F { s.A = <G*1 A s.A B>;  
  e.Other = e.Other }  
  
G*1 = { /* */ }
```



Вызов функции **G** с таким набором аргументов никогда не приведет к сопоставлению с левой частью предложения функции **G**. Вызов заменяется на вызов функции **G*1** для оптимизации других предложений

Результат решения уравнения – Undefined

В этом случае вызов оптимизирован не будет.

Изменение семантики функций

```
F {  
    e.A s.X e.B = <G s.X>;  
    e.A = Z;  
}  
  
$DRIVE G;  
  
G {  
    A = 1;  
    B = 2;  
}
```

```
G*2 {  
  
}  
  
F {  
    e.A A e.B = 1;  
    e.A B e.B = 2;  
    e.A s.X e.B = <G*2 s.X>  
    e.A = Z;  
}
```

При передаче F аргумента 'CA' до прогонки функция завершится с ошибкой, а после – вернет 1.

Левые части прогоняемых предложений должны быть *L-выражениями*: не должно быть открытых переменных и повторных t - и e -переменных.

Встраивание

Исключаются ситуации неоднозначного решения

```
$INLINE Apply;

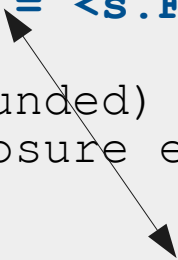
Apply {
  s.Fn e.Argument = <s.Fn e.Argument>;

  (t.Closure e.Bounded) e.Argument
    = <Apply t.Closure e.Bounded e.Argument>;
}

F { e.Args = <Apply Foo e.Args>}

/* после встраивания */

F { e.Args = <Foo e.Args>}
```



Прогонка функции `Apply` выполняется до предела итераций оптимизаций и только увеличивает размер правой части предложения, что приводит к ухудшению производительности.

Тестирование

Ряд функций из стандартной библиотеки были помечены, как встраиваемые. Все анонимные функции подвергаются прогонке по умолчанию, также некоторые функции для генерации кода были помечены, как прогоняемые.

Тестирование производилось для:

- Стандартной версии компилятора
- Сбранного с оптимизацией встраивания
- Сбранного с оптимизацией прогонки

Сборка компилятора в каждом случае производилась 17 раз. Для времени компиляции были измерены медиана и значения на границах первого и четвертого квартилей. Также в каждом из случаев было измерено количество шагов рефал-машины.

Тестирование

Сборка	Кол-во шагов	Время компиляции, медиана, с	Время компиляции, I квартиль, с	Время компиляции IV квартиль, с
Без оптимизаций	27607449	29.45	29.04	29.48
Оптимизация встраивания	25545258	26.23	26.22	26.25
Оптимизация прогонки	24683318	25.45	25.43	25.49

1. Встраивание: уменьшение кол-ва шагов - 2062191, уменьшение времени компиляции на 10.9%.
2. Прогонка: уменьшение кол-ва шагов - 2924131, уменьшение времени компиляции на 13.5%.

Заключение

В результате выполнения работы:

- Были разработаны алгоритмы оптимизации прогонки и встраивания.
- Алгоритмы оптимизации были успешно применены к самому компилятору и обеспечили прирост его производительности

Оптимизации прогонки и встраивания могут быть использованы и для других программ, написанных на РЕФАЛ-5λ.

Дальнейшее развитие:

- Улучшение алгоритмов оптимизации для более широкого подмножества языка РЕФАЛ-5λ.
- Обработка частных случаев при неизвестном решении уравнений сопоставления.
- Автоматическая разметка прогоняемых и встраиваемых функций.