Ордена Ленина
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
имени М.В. Келдыша
Академии наук СССР

С.А. Романенко

РЕФАЛ - 4 - РАСШИРЕНИЕ РЕФАЛА - 2, ОБЕСПЕЧИВАЮЩЕЕ ВЫРАЗИМОСТЬ РЕЗУЛЬТАТОВ ПРОГОНКИ

Препринт № 147 за 1987г.

Москва

Ордена Ленина
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
им. М.В.Келдыша
Академии Наук СССР

С.А.Романенко

РЕФАЛ-4 - РАСШИРЕНИЕ РЕФАЛА-2, ОБЕСПЕЧИВАЮЩЕЕ ВЫРАЗИМОСТЬ РЕЗУЛЬТАТОВ ПРОГОНКИ

Москва
1987

УДК 681.3.06

В работе описан ряд конструкций, которые предлагается ввести в Рефал-2. Эти конструкции позволяют выразить средствами входного языка рефал-системы результаты различных оптимизаций, выполняемых рефал-компилятором, но не выразимых средствами Рефала-2. Рефал-2, расширенный предлагаемыми конструкциями, именуется Рефалом-4. Результаты применения прогонки к программам на Рефале-4 выразимы средствами самого Рефала-4, что не имеет места для программ на Рефале-2. Изобразительные средства Рефала-4 являются более мощными, чем средства "Полного Рефала".

КЛЮЧЕВЫЕ СЛОВА И ФРАЗЫ: обработка символьной информации, преобразование программ, рекурсия, рефал, сопоставление с образцом, функциональное программирование.

С О Д Е Р Ж А Н И Е

В В Е Д Е Н И Е

В работах [ТУР 72], [ТУР 74] была предложена п р о г о н к а - система преобразований, обеспечивающая возможность выполнять метавычисления (в частности - частичные вычисления) для программ, написанных на языке Рефал [ТУР 66], [ТУР 71], [БЗР 77], [КР 87]. (Подобная система преобразований была позднее описана в [БД 77] для языка, являющегося по существу подмножеством "ограниченного рефала" [ТУР 72], [ТУР 74].) Прогонка является основой "суперкомпилятора" - системы автоматического анализа и преобразования рефал-программ [ТУР 86].

Ясно, что всякая система преобразований рефал-программ, предназначенная для практического применения, должна уметь обрабатывать реальные рефал-программы, написанные, например, на Базисном Рефале [БЗР 77] или на Рефале-2 [КР 87].

Между тем, прогонка в том виде, как она была известна до настоящего времени, применима только к программам, написанным на так называемом "ограниченном рефале" [ТУР 72], [ТУР 74]. Главной особенностью ограниченного рефала является то, что в нем запрещено употребление открытых VE-переменных, а также - повторных W- и VE-переменных.

Как представляется автору, прогонка была сформулирована только для ограниченного рефала не по той причине, что ее невозможно выполнять для Рефала-2, а в силу того, что ее результаты средствами самого Рефала-2 невыразимы.

Ясно, что удобно работать с такой системой преобразований, которая не выводит нас за рамки языка, на котором написаны программы. Другими словами, с таким языком, который является "неподвижной точкой" относительно данной системы преобразований. Получить "неподвижную точку" можно двумя способами: либо урезать язык, либо расширить его. Конечно, есть еще и третья возможность - отказаться от языка вообще, но мы ее рассматривать не будем.

Первая возможность - урезание языка - уже использована

[ТУР 72], [ТУР 74]. Результатом этой операции является ограниченный рефал. В данной работе рассматривается вторая возможность - расширение языка. Рефал-2 расширяется ровно до такой степени, чтобы получилась следующая "неподвижная точка", выдерживающая прогонку. Это расширение будет именоваться в дальнейшем Рефалом-4.

Конечно, с формально-логической точки зрения приведенные выше соображения противоречивы, ибо точно сформулировать саму систему преобразований можно только после того, как точно зафиксирован язык. Эти рассуждения, все же приобретают некоторый смысл, если мы попробуем истолковать понятие прогонки неформально, исходя не из технических деталей, а из преследуемых ею целей.

I. СОДЕРЖАТЕЛЬНАЯ СУЩНОСТЬ ПРОГОНКИ

Работа рефал-машины состоит из отдельных шагов. Каждый шаг распадается на две части - анализ (синтаксическое отождествление) и синтез (построение результата замены). Схематически это можно изобразить следующим образом:

A S A S A S ...

Анализ заключается в том, что аргумент вызова функции разлагается на несколько составных частей, которые затем, на этапе синтеза, перетасовываются в новом порядке и соединяются вновь в одно целое - результат замены. После этого начинается следующий шаг, и, довольно часто, только что построенные выражения тут же разбиваются снова, часто - на те же самые куски. Ясно, что в таких случаях эти куски можно было бы не соединять вовсе.

Такая перестройка выражений, происходящая на каждом шаге, может занимать значительное время. Кроме того, после завершения шага, вся информация об этих выражениях теряется, даже если она нужна при выполнении следующего шага. Это приводит к тому, что иногда один и тот же анализ повторяется несколько раз.

В подобных случаях выгодно укрупнять шаги рефал-машины.

Но ведь каждый шаг прогонки как раз и заключается в об'единении двух шагов рефал-машины в один, "более крупный" шаг!

Рассмотрим два соседних шага рефал-машины:

A' S' A'' S''

Эти два шага прогонка должна слить в один шаг: A S, где анализ A является в определенном смысле композицией анализов A' и A'', а синтез S - композицией синтезов S' и S''. Таким образом, преобразованная программа должна уметь выполнять анализ A', а вслед за ним - анализ A''. Но ведь в исходной программе A' и A'' разделены синтезом S'! В этом и заключаются основные трудности при выполнении прогонки.

Таким образом, основная задача, с которой должна справляться прогонка - это продолжать анализ, не делая промежуточного синтеза.

Ниже будет описано расширение Рефала-2, которое оказывается "неподвижной точкой" по отношению к прогонке.

2. ПЕРЕХОД К БЕЗБЛАНКОВОЙ ФОРМЕ ЗАПИСИ ПРОГРАММ

При расширении рефала возникает необходимость во вложенных синтаксических конструкциях, поэтому первым нашим шагом будет переход от бланковой формы записи программ, к безбланковой форме, в которой границы между записями не имеют никакого значения. А именно, описание функции, которое раньше записывалось в виде

```
F       L1 = R1
        L2 = R2
         ...
        LN = RN
```

будет выглядеть следующим образом:

```
#FUN  F
   #     L1 = R1
   #     L2 = R2
          ...
   #     LN = RN
#END  F
```

где $L\dot{I}$ - левые части предложений, а $R\dot{I}$ - соответствующие правые части предложений. Каждая левая часть представляет собой о б р а з е ц, т.е. типовое выражение, перед которым может стоять указатель направления отождествления: #L или #R.

3. ПЕРЕСТРОЙКИ

Одним из основных изобразительных средств, которые мы введем в рефал будут п е р е с т р о й к и. Каждая перестройка имеет следующий вид:

[ИСТОЧНИК] : [ПОЛУЧАТЕЛЬ]

Источником является выражение (в смысле Рефала-2), а получателем - образец, т.е. типовое выражение, перед которым может стоять указатель направления отождествления #L или #R. Например:

```
EA : #R E1 '+' E2
EA EB : EX
<F EA> 'A' : S1 S2
```

Семантика перестройки определяется следующим образом. В момент исполнения перестройки некоторые из переменных, входящих в описание функции, уже имеют значение, а некоторые - еще не имеют. В источник перестройки могут входить только такие переменные, которые к моменту ее исполнения уже имеют значение, в получатель могут входить любые переменные. Первый шаг исполнения перестройки заключается в том, что вместо всех переменных, уже имеющих значения, в перестройку подставляются их значения. При этом источник превращается в выражение, не содержащее переменных. Если это выражение содержит вызовы функций, эти вызовы вычисляются. Если при вычислении этих вызовов возникает аварийный останов "отождествление невозможно", то работа всей рефал-программы аварийно завершается. Если же эти вычисления проходят успешно, то перестройка принимает вид

[ИСТОЧНИК'] : [ПОЛУЧАТЕЛЬ']

где [ИСТОЧНИК'] является об'ектным выражением, а [ПОЛУЧАТЕЛЬ'] содержит только переменные, еще не получившие значение.

После этого выполняется синтаксическое отождествление: источник, принятым в Рефале-2 способом, сопоставляется с образцом. При этом, переменные, входящие в образец, получают значение. При отождествлении принимается во внимание указатель направления отождествления. Отсутствие указателя направления означает отождествление слева направо.

Попытка сопоставления с образцом может либо потерпеть неудачу (если отождествление невозможно), либо пройти успешно. Соответственно считается, что и вся перестройка либо прошла успешно, либо потерпела неудачу.

Если получатель содержит открытые вхождения VE-переменных, может возникнуть неоднозначность при сопоставлении с образцом. В этом случае способы отождествления упорядочиваются, как это принято в Рефале-2, а переменные образца получают значения в соответствии с первым способом отождествления. Затем информация о текущем состоянии рефал-машины заносится в с т е к а л ь т е р н а т и в и

выполнение программы продолжается. Если при дальнейшем исполнении какая-то часть программы терпит неудачу, из стека альтернатив извлекается информация о самом последнем месте, где возникала неоднозначность. Если эта неоднозначность возникла в перестройке, то переменные из образца этой перестройки получают значения, которые соответствуют следующему по порядку способу отождествления и выполнение программы повторяется снова с этой точки. Если следующего способа отождествления не существует, перестройка терпит неудачу.

Для некоторых частных случаев перестройки будем использовать особые названия.

Перестройки, имеющие источник, состоящий из одной-единственной переменной, будут называться с у ж е н и я м и. При исполнении сужения не требуется синтез выражений. Например, если переменная EA уже имеет значение, а переменные EX и EY - еще не имеют, то перестройка

```
EA : #R EY '+' EX
```

является сужением.

Перестройки, у которых получатель состоит из одной-единственной переменной, еще не имеющей значения, а источник порождает только такие выражения, которые заведомо отождествимы с получателем, будут называться п р и с в а и в а н и я м и. При исполнении присваивания не требуется делать анализ выражений. Например, если переменные EA, EB и SC же имеют значение, а переменная VX - не имеет, то перестройка

```
EA SC EB : VX
```

является присваиванием.

"Правые части" предложений, имеющие вид

```
= R
```

в некотором смысле являются присваиваниями. А именно, если обозначить результат вызова функции через [RES], то правые

части можно трактовать как присваивания вида

```
R : [RES]
```

с последующим выходом из функции. Поэтому в дальнейшем они будут именоваться в ы х о д н ы м и присваиваниями.

"Левые части" предложений, имеющие вид

```
#  L
```

в некотором смысле являются сужениями. А именнно, если обозначить аргумент функции через [ARG], то левые части можно трактовать как сужения вида

```
[ARG] : L
```

Поэтому в дальнейшем они будут именоваться в х о д н ы м и сужениями.

Перестройки являются обобщением понятий сужения и присваивания [ТУР 86]. В то же время, они имеют определенное сходство с инструкциями замещения Снобола-4 [ГПШ 80].

4. ИСПОЛЬЗОВАНИЕ ПЕРЕСТРОЕК

Разрешим использовать перестройки в рефал-программах. А именно, вместо предложений вида

```
#  L = R
```

разрешим писать предложения вида

```
#  L, C1, C2, ..., CN = R
```

где C1, C2,... , CN - перестройки. Эти перестройки должны исполняться в том порядке, как они написаны, после сопоставления аргумента с L.

Например, вместо

```
#  (SX EA)(SY EB) = (SX EA) (SY EB)
```

можно написать

```
#   E1, E1 : (E2)(E3),
        E2 : SX EA,
        E3 : SY EB      = E1
```

а вместо

```
#   #L EX '+' EY '*' EZ = (EX)(EY)(EZ)
```

можно написать

```
#   #L EX '+' E1,   E1 : #L EY '*' EZ
                              = (EX)(EY)(EZ)
```

Сужения позволяют более гибко управлять процессом отождествления. Пусть, например, требуется найти в выражении самый первый и самый последний '+'. Если программировать это на Рефале-2, то потребуется использовать два шага рефал-машины, ибо ни отождествление слева направо, ни отождествление справа налево не решают проблемы. Однако, с помощью сужений это можно сделать следующим образом:

```
E1 : #L EX '+' E1,   E1 : #R EZ '+' EY
```

5. ВЕТВЛЕНИЯ

Теперь введем еще одну конструкцию, которая позволит нам, вкупе с сужениями, об'единять совпадающие части образцов. Эта управляющая конструкция имеет вид:

```
#ALT
   #PATH A1
   #PATH A2
     ...
```

```
    #PATH AN
  #END
```

где Aİ - фрагмент программы, который может содержать перестройки, выходные присваивания и вложенные конструкции #ALT.

Исполняется #ALT следующим образом. Когда управление попадает на эту конструкцию, в стек альтернатив заносится информация о текущем состоянии рефал-машины и выполняется ветвь A1. Если A1 терпит неудачу, то восстанавливается то состояние рефал-машины, которое было при входе в #ALT и делается попытка выполнить ветвь A2, и т.д. Если попытка пройти через ветвь AN окончилась неудачей, то и вся конструкция #ALT терпит неудачу.

Теперь осталось рассмотреть что происходит, когда удается успешно дойти до конца ветви Aİ (не потерпев неудачу и не встретив выходное присваивание). В этом случае управление просто передается на конструкцию, следующую за закрывающим #END. В стеке альтернатив при этом не делается никаких изменений. Таким образом, если последующая часть программы терпит неудачу, то может произойти возврат внутрь ветви Aİ. Например:

```
#ALT
  #PATH  EA : #L E1 SX E2,    SX : EZ
  #PATH  EA : #R E1 (EY) E2, EY : EZ
#END
```

Здесь, в процессе подбора способа отождествления, сначала рассматриваются все символы, стоящие на нулевом уровне выражения EA, а затем - все выражения, стоящие на первом уровне выражения EA.

Возникает вопрос: какие переменные считаются имеющими значение после выхода из #ALT? Ведь на различных ветвях могут получать значение различные переменные! Мы будем придерживаться следующего соглашения: после выхода из #ALT будут иметь значение только те переменные, которые имеют значение в конце к а ж д о й из ветвей, составляющих #ALT. Так, в только что рассмотренном примере переменная EZ будет

иметь значение после выхода из #ALT, а переменные SX и EY - не будут иметь значения.

Таким образом, мы видим, что конструкция #FUN в неявном виде содержит внутри себя конструкцию #ALT. А именно, если обозначить через [ARG] аргумент обращения к функции, то

```
#FUN F
  #  LI AI
  #  L2 A2
     ...
  #  LN AN
#END F
```

в некотором смысле эквивалентно

```
#ALT
   #PATH [ARG] : LI AI
   #PATH [ARG] : L2 A2
     ...
   #PATH [ARG] : LN AN
#END
```

Следует заметить, что выходное присваивание вида "= R" означает, что выражение R следует вычислить и выдать в качестве результата функции. Поэтому управление никогда не попадает на конструкции, следующие непосредственно за "= R".

Конструкция #ALT имеет определенное сходство с конструкцией ALT языка Плэнер [ПИЛ 83].

Пользуясь конструкцией #ALT мы можем строить дерево из совпадающих частей образцов. Например, функция

```
#FUN F
   #    'AA' EX = '1'
   #    'AB' EX = '2'
   #    'BA' EX = '3'
   #    'BB' EX = '4'
#END F
```

может быть переписана следующим образом:

```
#FUN F
  # 'A' E1  #ALT
              #PATH  E1 : 'A' EX = '1'
              #PATH  E1 : 'B' EX = '2'
             #END
  # 'B' E1  #ALT
              #PATH  E1 : 'A' EX = '3'
              #PATH  E1 : 'B' EX = '4'
             #END
#END F
```

Таким образом, средствами Рефала-4 выразим результат оптимизации по об'единению совпадающих частей образцов. Эта оптимизация выполняется многими компиляторами с рефала на язык сборки, но невыразима средствами Рефала-2 или Базисного Рефала [КРТ 72], [БЗР 77].

6. НЕУСПЕХ ПРИ ИСПОЛНЕНИИ ПРОГРАММЫ

В некоторых случаях возникает желание указать в явном виде, что отождествление в данном месте зашло в тупик. Для этого используется конструкция

```
#FAIL
```

#FAIL по своему действию эквивалентно перестройке, исполнение которой заведомо терпит неудачу, например перестройке

```
'A' : 'B'
```

С другой стороны #FAIL эквивалентно конструкции #ALT с нулевым числом ветвей:

```
#ALT #END
```

В некоторых случаях требуется указать, что продолжение исполнения рефал-программы невозможно, ибо обрабатываемые выражения имеют недопустимую (или непредусмотренную) структуру. Для этого используется конструкция

#ABORT

Ее исполнение приводит к аварийному останову программы "отождествление невозможно".

7. УПРАВЛЕНИЕ ПЕРЕХОДАМИ

Для управления перебором средствами входного языка используются две конструкции #GATE (ворота) и #FENCE (забор).

Исполнение конструкции #GATE приводит к тому, что текущее состояние стека альтернатив заносится в стек состояний, а стек альтернатив становится пустым.

Исполнение конструкции #FENCE сводится к тому, что текущее содержимое стека альтернатив уничтожается. Затем из стека состояний извлекается последний элемент и заносится в стек альтернатив.

Если исполнение какой-либо конструкции терпит неудачу, а стек альтернатив пуст, то из стека состояний извлекается последний элемент и заносится в стек альтернатив.

Таким образом, можно свободно проходить через "ворота" и в прямом, и в обратном направлении. Через "забор" можно перепрыгивать при движении слева направо. Однако, если мы натыкаемся на забор при возврате из-за неудачи, то проходить сквозь забор нельзя: вместо этого следует перескочить на соответствующие ему ворота (Рис.I).

В начале исполнения вызова функции в стек состояний заносится пустой элемент, а стек альтернатив очищается. Таким образом, конструкция #FUN подразумевает не просто #ALT, а #GATE #ALT.

Вернемся теперь к построению дерева из совпадающих частей образцов. В рассмотренном выше примере после успешного

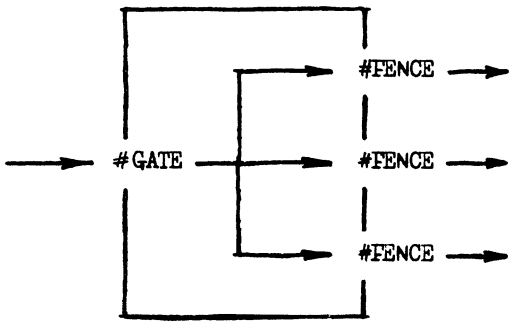

Рис.I. Управление переходами с помощью #GATE и #FENCE.

выполнения сужения [ARG] : 'A' EI и неуспеха последующих сужений нет смысла перепрыгивать на следующую ветвь и пробовать сужение [ARG] : 'B' EI. Поэтому описание функции можно усовершенствовать, отменив ненужные возвраты.

```
#FUN F
  #   'A' EI #FENCE
            #ALT
              #PATH EI : 'A' EX = 'I'
              #PATH EI : 'B' EX = '2'
            #END
  #   'B' EI #FENCE
            #ALT
              #PATH EI : 'A' EX = '3'
              #PATH EI : 'B' EX = '4'
            #END
#END F
```

Используя #GATE и #FENCE можно явно записать результат оптимизаций по сокращению перебора значений VE-переменных. Например, в случае сужения

```
EA : #L EX '+' EY '*' EZ
```

если терпит неудачу попытка удлинить значение переменной EY, то нет смысла пытаться удлинять значение переменной EX. Поэтому данное сужение можно разбить на два сужения и ограничить перебор:

```
#GATE
   EA : #L EX '+' EI
   #FENCE
EI : #L EY '*' EZ
```

Таким образом, средствами Рефала-4 выразимы результаты оптимизаций по сокращению перебора при отождествлении. Эти оптимизации выполняются многими компиляторами с рефала на язык сборки, но невыразимы средствами Рефала-2 или Базисного Рефала [КРТ 72], [БЗР 77].

8. СИНТАКСИС ОПРЕДЕЛЕНИЯ ФУНКЦИИ

В результате введения в Рефал-2 конструкций, описанных выше, синтаксис определения функции принимает следующий вид:

```
[определение-функции] ::=
  #FUN [имя-функции] [предложения] #END [имя-функции]
[предложения] ::=
   [пусто] | [предложение] [предложения]
[предложение] ::=
  # [входное-сужение] [отделенная-ветвь]
[ветвь] ::=
   [пусто] |
   [перестройка] [отделенная-ветвь] |
   [выходное-присваивание] [отделенная-ветвь] |
   [управление] [ветвь]
[отделенная-ветвь] ::=
   [пусто] |
   [выходное-присваивание] [отделенная-ветвь] |
```

```
    [управление]  [ветвь]
[перестройка] ::=
    [источник] : [получатель]
[входное-сужение] ::=
    [получатель]
[выходное-присваивание] ::=
    = [источник]
[источник] ::=
    [выражение]
[получатель] ::=
    [образец]
[управление] ::=
    [пустое-действие] |
    #GATE | #FENCE | #FAIL | #ABORT |
    [распутье]
[пустое-действие] ::=

[распутье] ::=
    #ALT  [тропы]  #END
[тропы] ::=
    [пусто] | [тропа]  [тропы]
[тропа] ::=
    #PATH  [ветвь]
[образец] ::=
    [указатель-направления]  [типовое-выражение]
[указатель-направления] ::=
    [пусто] | #L | #R
[пусто] ::=
```

9. РЕАЛИЗАЦИЯ ПОЛНОГО РЕФАЛА ЧЕРЕЗ ПЕРЕСТРОЙКИ

В [ТУР 71] было описано расширение рефала под названием "Полный Рефал", аналогичный проект можно найти в [БЕЛ 85]. Используя перестройки, источники которых содержат вызовы функций, мы можем перемежать синтаксическое отождествление с вычислением вызовов функций. Благодаря этому, рассматриваемое здесь расширение рефала является более мощным по

выразительным возможностям, чем "Полный Рефал".

Например, пусть синтаксис идентификаторов определен следующим образом:

```
[ID] ::= [LETTER] |
         [ID] [DIGIT] |
         [ID] [LETTER]
```

Пусть функции "LETTER" и "DIGIT" вырабатывают 'T', если их аргумент - буква или цифра, соответственно, и вырабатывают 'F' в противном случае.

Определим функции "GET-LETTER" и "GET-DIGIT" следующим образом:

```
#FUN GET-LETTER
 #   EI SX, <LETTER SX>:'T' = EI (SX)
 #   EI = EI '*'
#END GET-LETTER
#FUN GET-DIGIT
 #   EI SX, <DIGIT  SX>:'T' = EI (SX)
 #   EI = EI '*'
#END GET-DIGIT
```

Тогда функцию-предикат, проверяющую, что цепочка литер является идентификатором, можно записать следующим образом:

```
#FUN ID #  EA
   #GATE #ALT
     #PATH  <GET-LETTER EA> : E2 (SI) #FENCE
         #ALT
           #PATH  E2 :  = 'T'
           #PATH        = <ID E2>
         #END
     #PATH  <GET-DIGIT  EA> : E2 (SI) #FENCE = <ID E2>
     #PATH                            #FENCE = 'F'
   #END
#END ID
```

Конечно, это описание нарочито уродливо, зато оно показывает, какой простор открывается для творчества тех, кто захочет выжать из перестроек все возможности.

I0. ПРОГОНКА

Каждый шаг прогонки, как он описан в [ТУР 72], [ТУР 74], представляет собой довольно сложное и не расчлененное на более простые этапы преобразование рефал-программы. Изобразительные средства, имеющиеся в Рефале-4, позволяют расчленить шаг прогонки в последовательность более простых преобразований. А именно, шаг прогонки разделяется на символическое выполнение вызова функции (раскрытие функционального терма) и преобразование перестроек общего вида в последовательность сужений, за которой следует последовательность присваиваний.

Раскрытием функционального терма достигается формальная цель прогонки (соединение двух шагов рефал-машины в один), а преобразованием перестроек достигается содержательная цель прогонки (соединение двух отождествлений в одно без промежуточного построения выражений).

В данной работе мы рассмотрим только первую часть прогонки - раскрытие функциональных термов.

В дальнейшем, чтобы не загромождать изложение, будем предполагать, что все функции в программе, над которой выполняются преобразования, являются именными, т.е. их вызов не имеет побочных эффектов. Обобщение прогонки на случай, когда имеются вызовы глагольно-именных функций (дающих побочный эффект), не создает никаких особых затруднений.

При описании прогонки рассмотрим два различных случая: раскрытие функционального терма, стоящего в источнике выходного присваивания, и раскрытие функционального терма, стоящего в источнике обычной перестройки.

В раскрытии функционального терма участвуют две функции: функция, в описании которой находится раскрываемый функциональный терм, и функция, к которой этот функциональный терм обращается. Пусть эти функции имеют имена "F" и "G"

соответственно.

Пусть описание функции "G" имеет вид:

```
#FUN G
  #  L1 A1
  #  L2 A2
     ...
  #  LN AN
#END G
```

Будем предполагать, что переменные, входящие в это описание, имеют индексы, которые отличаются от индексов переменных, входящих в описание функции "F". Если это условие не выполнено, следует предварительно выполнить переименование переменных, входящих в описание "G".

Кроме этого, будем считать, что все #GATE-ы, входящие в описание функций "F" и "G", закрыты с помощью #FENCE-ов в явном виде. Если это не так, следует перед выходными присваиваниями вставить достаточное количестов #FENCE-ов. Напоминаем, что #FUN содержит внутри себя один #GATE в неявном виде. Он тоже должен быть закрыт #FENCE-ом.

Пусть теперь раскрываемый терм вида <G E> входит в выходное присваивание

```
= C1 <G E> C2
```

Тогда раскрытие терма заключается в том, что это выходное присваивание заменяется на конструкцию

```
#GATE #ALT
   #PATH E : L1 B1
   #PATH E : L2 B2
     ...
   #PATH E : LN BN
#END
```

где Bi получается из Ai заменой каждого выходного присваивания вида "= R", входящего в Ai, на выходное присваивание

```
= C1 R C2
```

Теперь рассмотрим случай, когда раскрываемый терм входит в перестройку вида

```
C1 <G E> C2 : L
```

Теперь раскрытие терма заключается в том, что эта перестройка заменяется на конструкцию

```
#GATE #ALT
   #PATH E : L1 B1
   #PATH E : L2 B2
    ...
   #PATH E : LN BN
   #PATH #ABORT
#END
```

где Bi получается из Ai заменой каждого выходного присваивания вида "= R" на перестройку

```
, C1 R C2 : L
```

Например, пусть функции "F" и "REV" имеют следующие описания:

```
#FUN REV
  #          #FENCE =
  #  SX EY   #FENCE = <REV EY> SX
#END REV
#FUN F
  #  EA  #FENCE = <REV EA>
#END F
```

Раскроем терм "<REV EA>" в описании функции "F". Получаем

```
#FUN F
  #  EA #FENCE
      #GATE #ALT
         #PATH  EA :          #FENCE =
         #PATH  EA : S1 E2    #FENCE = <REV E2> S1
      #END
#END F
```

Теперь раскрываем терм "<REV E2>" в преобразованном описании функции "F". Получаем

```
#FUN F #  EA  #FENCE
  #GATE #ALT
    #PATH  EA :          #FENCE =
    #PATH  EA : S1 E2    #FENCE
       #GATE #ALT
          #PATH  E2 :          #FENCE = S1
          #PATH  E2 : S3 E4    #FENCE = <REV E4> S3 S1
       #END
  #END
#END F
```

Теперь рассмотрим пример прогонки в случае, когда раскрываемый терм находится в источнике перестройки. Пусть описание функции "F1" имеет вид

```
#FUN F1 # EX  #FENCE
   #ALT
     #PATH  <REV EX>: EX  #FENCE = 'T'
     #PATH                #FENCE = 'F'
   #END
#END F1
```

Тогда раскрытие обращения к "REV" дает

```
#FUN FI  # EX  #FENCE
  #ALT
    #PATH
      #GATE #ALT
          #PATH  EX :        #FENCE                : EX
          #PATH  EX : SA EB #FENCE  <REV EB> SA : EX
          #PATH  #ABORT
       #END      #FENCE = 'T'
     #PATH       #FENCE = 'F'
  #END
#END FI
```

Теперь рассмотрим пример, в котором существенное место занимают открытые вхождения VE-переменных. Раньше такие вхождения были неодолимым препятствием для прогонки. Пусть функции "F" и "G" описаны следующим образом:

```
#FUN F
   #  #L EX '+' EY  #FENCE = (EX)'+' <G EY>
   #  #L EX '-' EY  #FENCE = (EX)'-' <G EY>
#END F
#FUN G
   #  #R EA 'A' EB  #FENCE = (EA)'A'(EB)
   #  #R EA 'B' EB  #FENCE = (EA)'B'(EB)
#END G
```

Теперь раскроем по одному разу обращения к "G" в описании функции "F". Получается

```
#FUN F
 #   #L EX '+' EY  #FENCE
     #GATE #ALT
        #PATH  EY : #R EA 'A' EB  #FENCE
                                  = (EX)'+'(EA)'A'(EB)
        #PATH  EY : #R EA 'B' EB  #FENCE
                                  = (EX)'+'(EA)'B'(EB)
     #END
 #   #L EX '-' EY  #FENCE
     #GATE #ALT
        #PATH  EY : #R EA 'A' EB  #FENCE
                                  = (EX)'-'(EA)'A'(EB)
        #PATH  EY : #R EA 'B' EB  #FENCE
                                  = (EX)'-'(EA)'B'(EB)
     #END
#END F
```

З А К Л Ю Ч Е Н И Е

Язык Рефал-4, который получается из Рефала-2 введением в последний ряда конструкций, отсутствовавших в Базисном Рефале [БЗР 77] и в Рефале-2 [КР 87], обладает рядом привлекательных свойств.

* Средствами Рефала-4 выразимы результаты ряда оптимизаций, выполняемых компиляторами с рефала на язык сборки [БЗР 77], [КРТ 72], [РКТ 73]. Эти оптимизации были невыразимы средствами Рефала-2 и, тем более, Базисного Рефала.

* Рефал-4 предоставляет более мощные изобразительные средства, чем "Полный Рефал" [ТУР 71].

* Результаты применения прогонки к программам на Рефале-4 выразимы (как и в случае "Ограниченного Рефала" [ТУР 72], [ТУР 74]), средствами самого Рефала-4. Это не имеет места для программ на Базисном Рефале и Рефале-2. В этом смысле Рефал-4, как и "Ограниченный Рефал", является "неподвижной точкой" относительно прогонки. При этом, в отличие от "Ограниченного Рефала", он является не подмножеством Базисного Рефала и Рефала-2, а их расширением.

В заключение следует отметить, что в данной работе рассмотрено только одно преобразование программ на Рефале-4 - раскрытие функциональных термов. Посредством этого преобразования достигается формальная цель прогонки - соединение двух шагов рефал-машины в один. Содержательная цель прогонки - соединение двух актов отождествления в один - может быть достигнута с помощью ряда преобразований перестроек. Этот вопрос, однако, не рассматривался в данной работе.

Л И Т Е Р А Т У Р А

[БД 77]
R.M.BURSTALL, J.DARLINGTON. A TRANSFORMATION SYSTEM FOR DEVELOPING RECURSIVE PROGRAMS. J.ACM 24 (1977), 44-67.

[БЕЛ 85]
А.П.Бельтюков. Эффективное расширение языка Рефал. - Вычислительные методы и информационное обеспечение пакетов прикладных программ. Сборник научных трудов. Устинов, 1985, с.50-54.

[БЗР 77]
Базисный рефал и его реализация на вычислительных машинах. М., ЦНИПИАСС, 1977, с.92-95.

[ГШ 80]
Р.Грисуолд, Дж.Поудж, И.Полонски. Язык программирования Снобол-4. М.: Мир, 1980. - 269 с.

[КР 86]
Ан.В.Климов, С.А.Романенко. Система программирования Рефал-2 для ЕС ЭВМ. Описание библиотеки функций. Препринт ИПМ им.М.В.Келдыша АН СССР, М., 1986, N 200.

[КР 87]
Ан.В.Климов, С.А.Романенко. Система программирования Рефал-2 для ЕС ЭВМ. Описание входного языка. ИПМ им.М.В.Келдыша АН СССР, М., 1987.

[КРТ 72]
Ан.В.Климов, С.А.Романенко, В.Ф.Турчин. Компилятор с языка рефал. ИПМ АН СССР, М., 1972.

[ПИЛ 83]

В.Н.Пильщиков. Язык плэнер. М.: Наука. Главная редакция физико-математической литературы, 1983. - 208 с.

[РКТ 73]

С.А.Романенко, Ан.В.Климов, В.Ф.Турчин. Теоретические основы синтаксического отождествления в языке рефал. Препринт ИПМ АН СССР N 13, М., 1973.

[ТУР 66]

В.Ф.Турчин. Метаязык для формального описания алгоритмических языков. - В сб.: Цифровая вычислительная техника и программирование, М.: Сов. радио, 1966, с.116-124.

[ТУР 71]

В.Ф.Турчин. Программирование на языке Рефал. Препринты ИПМ АН СССР N 41, 43, 44, 48, 49. М., 1971.

[ТУР 72]

В.Ф.Турчин. Эквивалентные преобразования рекурсивных функций, описанных на языке Рефал. - В сб.: Теория языков и методы построения систем программирования. Труды симпозиума, Киев-Алушта: 1972, с. 31-42.

[ТУР 74]

В.Ф.Турчин. Эквивалентные преобразования программ на Рефале. - В сб.: Автоматизированная система управления строительством. М.: ЦНИПИАСС, 1974, вып.4, с.36-68.

[ТУР 86]

V.F.TURCHIN. THE CONCEPT OF A SUPERCOMPILER. ACM TRANSACTIONS ON PROGRAMMING LANGUAGES AND SYSTEMS, VOL.8, NO.3, JULY 1986, PP.292-325.

С.А. Романенко " Рефал - 4 - расширение Рефала - 2, обеспечива - ющее выразимость результатов прогонки."
Редактор Вс.С. Штаркман. Корректор А.В. Климов.

Подписано к печати 6.07.87г. № Т- 16120. Заказ № 299.
Формат бумаги 60X90 1/16. Тираж 200 экз.
Объём 1,2 уч.-изд.л. Цена 15 коп.

055 (02)2

Отпечатано на ротапринтах в Институте прикладной математики АН СССР
Москва, Миусская пл. 4.

Ссылки на издание рекомендуется делать по следующей форме: и.о., фамилия, название, препринт Ин. прикл. матем. им. М.В. Келдыша АН СССР, год, №.

Распространение: препринты института продаются в магазинах Академкниги г. Москвы, а также распространяются через Библиотеку АН СССР в порядке обмена.

Адрес: СССР, 125047, Москва-47, Миусская пл. 4, Институт прикладной математики им. М.В. Келдыша АН СССР, ОНТИ.

The references should be typed by the following form: initials, name, title, preprint, Inst.Appl.Mathem., the USSR Academy of Sciences, year, N(number).

Distribution. The preprints of the Keldysh Institute of Applied Mathematics, the USSR Academy of Sciences are sold in the bookstores "Academkniga", Moscow and are distributed by the USSR Academy of Sciences Library as an exchange.

Adress: USSR, I25047, Moscow A-47, Miusskaya Sq.4, the Keldysh Institute of Applied Mathematics, Ac.of Sc., the USSR, Information Bureau.

Цена 15 коп.