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

С.А. Романенко

ГЕНЕРАТОР КОМПИЛЯТОРОВ, ПОРОЖДЕННЫЙ САМОПРИМЕНЕНИЕМ СПЕЦИАЛИЗАТОРА, МОЖЕТ ИМЕТЬ ЯСНУЮ И ЕСТЕСТВЕННУЮ СТРУКТУРУ

Препринт № 26 за 1987 г.

Москва

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

С.А.Романенко

ГЕНЕРАТОР КОМПИЛЯТОРОВ, ПОРОЖДЕННЫЙ САМОПРИМЕНЕНИЕМ СПЕЦИАЛИЗАТОРА, МОЖЕТ ИМЕТЬ ЯСНУЮ И ЕСТЕСТВЕННУЮ СТРУКТУРУ

Москва
1987

УДК 681.3.06

Описана структура и принципы работы самоприменимого специализатора программ, а также принципы работы генератора компиляторов, автоматически порожденного самоприменением специализатора. Показано, что структура порождаемых компиляторов улучшается, если использовать различное представление для значений K- и U- переменных, разделение программы на K- и U-функции, а также автоматическое повышение арности функций.

КЛЮЧЕВЫЕ СЛОВА И ФРАЗЫ: генератор компиляторов, самоприменение специализатора, специализатор, специализация программ, частичные вычисления.

С О Д Е Р Ж А Н И Е

В В Е Д Е Н И Е

Пусть SPEC – функция от двух переменных, заданная программой и удовлетворяющая соотношению F(X,Y)=SPEC(F,X)(Y), где F – произвольная функция от двух переменных, заданная программой, а X и Y – входные данные для F. Такую функцию SPEC мы будем в дальнейшем именовать "специализатором". То, что специализаторы могут стать практически важным средством программирования, было осознано еще в шестидесятые годы [ЛОМ 67]. К началу семидесятых годов относится, видимо, появление самого термина специализатор [ДИК 71], [ЧНЛ 73].

В 1971 году Й.Футамура обнаружил, что компиляция программ может осуществляться специализацией интерпретаторов [ФУТ 71]. А именно, пусть INT – интерпретатор некоторого языка, т.е. функция от двух переменных, такая, что INT(S,D) = S(D), где через S(D) обозначен результат применения программы S к исходным данным D. Тогда, согласно [ФУТ 71], SPEC(INT,S) можно считать результатом компиляции программы S на выходной язык специализатора, ибо SPEC(INT,S)(D) = INT(S,D) = S(D).

В той же работе [ФУТ 71] Й.Футамура указал, что если входной и выходной языки специализатора совпадают и, таким образом, возможно самоприменение специализатора, то специализацией специализатора на интерпретатор можно преобразовать интерпретатор в компилятор, ибо SPEC(SPEC,INT)(S) = SPEC(INT,S).

В последующие годы [БЕК 76], [ТУР 77,79,80] было обнаружено, что двойным самоприменением специализатора может быть вычислен генератор компиляторов SPEC(SPEC,SPEC),

представляющий собой преобразователь интерпретаторов в компиляторы, ибо SPEC(SPEC,SPEC)(INT) = SPEC(SPEC,INT).

К сожалению, в течение ряда лет метод получения генератора компиляторов вычислением SPEC(SPEC,SPEC) казался хотя и заманчивой, но чисто теоретической возможностью. Хотя в литературе и можно найти сообщения о создании специализаторов, способных компилировать программы специализацией интерпретаторов [ФУТ 71], [ТУР 79], [ТНТ 82], ТУР 86], только в последнее время копенгагенской группе, возглавляемой Н.Джонсом, удалось создать нетривиальный специализатор, самоприменимый не только теоретически, но и практически [ДСЗ 85А, ДСЗ 85Б, СЕС 86], благодаря чему впервые удалось осуществить эксперименты по автоматическому преобразованию интерпретаторов в компиляторы по методу Й.Футамуры и даже вычислить нетривиальный генератор компиляторов SPEC(SPEC,SPEC).

В дальнейшем, для краткости, специализатор и генератор компиляторов, созданные группой под руководством Н.Джонса, мы будем именовать "копенгагенскими".

Следует отметить, что копенгагенский специализатор не является полностью автоматическим. В программу, подлежащую специализации, человеком должны быть предварительно внесены дополнительные указания. А именно, все вызовы функций в программе должны быть расклассифицированы на два класса: на "устранимые" и "остаточные" вызовы. Все устранимые вызовы выполняются во время специализации программы, а остаточные вызовы переносятся в результирующую программу и, таким образом, откладываются до момента исполнения специализированной программы. Аннотирование программы заключается в том, что для всех остаточных вызовов ключевое слово CALL заменяется на ключевое слово CALLR.

Проблема классификации вызовов на остаточные и устранимые, как оказалось, весьма трудна для компьютера. Сложность этой проблемы, видимо, является основной причиной, по

которой не удалось осуществить самоприменение полностью автоматических (и при этом - нетривиальных) специализаторов [ТУР 86].

Авторы [ДСЗ 85А], следующим образом характеризуют результаты, полученные с помощью копенгагенского специализатора.

Компиляторы, построенные вычислением SPEC(SPEC,INT), "имеют удивительно естественную структуру" ("HAVE A SURPRISINGLY NATURAL STRUCTURE"), однако с автоматически порожденным генератором компиляторов дело обстоит не так благополучно. Хотя он и "имеет приемлемые размеры", ("IS OF REASONABLE SIZE"), "разобраться в его логике труднее", ("ITS LOGIC IS HARDER TO FOLLOW"), чем в логике порожденных компиляторов, а в его тексте встречаются "неожиданные конструкции вроде ''''NIL" ("SOME UNEXPECTED CONSTRUCTIONS (LIKE ''''NIL)").

Получив, благодаря любезности Н.Джонса, в начале 1986 года подробную информацию о структуре и принципах работы копенгагенского специализатора, автор поставил перед собой задачу воспроизвести результаты, полученные копенгагенской группой. При этом, первоначальные цели автора состояли в следующем:

- попытаться применить копенгагенскую методику к программам, написанным на языке Рефал, а не на языке Лисп;

- увидеть воочию сгенерированные автоматически компиляторы и, в особенности, генератор компиляторов и изучить их структуру.

Эти цели, благодаря наличию подробной информации о копенгагенском специализаторе, были достигнуты без особых затруднений. Однако, получавшиеся компиляторы и генератор компиляторов оказались, на взгляд автора, эстетически неудовлетворительными. В компиляторах еще можно было

разобраться, а вот генератор компиляторов имел такой отталкивающий вид, что отбивал всякую охоту пытаться его прочитать.

Тем не менее, желание постичь загадочные принципы работы генератора компиляторов заставило автора искать способы улучшить структуру программ, выдаваемых специализатором. В результате, первоначальный вариант специализатора был переработан. Получившаяся версия будет в дальнейшем, для краткости, именоваться "московским специализатором".

Обсуждению общих черт и различий между копенгагенским и московским специализаторами и посвящены несколько последующих разделов работы. В заключение описывается структура и принципы работы автоматически сгенерированного генератора компиляторов.

I. ОСНОВНЫЕ ПРИНЦИПЫ РАБОТЫ ДВУХ СПЕЦИАЛИЗАТОРОВ

Основной особенностью копенгагенского специализатора, отличающей его от других, построенных ранее и основанных на частичных вычислениях, является то, что процесс специализации разделен на две фазы.

Во время первой фазы исходная программа подвергается абстрактной интерпретации над областью значений, состоящей только из двух элементов: символов "K" и "U". "K" изображает значение, которое будет известно частичному вычислителю в процессе построения остаточной программы, а "U" изображает значение, неизвестное во время частичных вычислений.

Поскольку частичные вычисления, по отношению к обычным, являются мета-вычислениями, а KU-интерпретация является мета-вычислением по отношению к частичным вычислениям, KU-интерпретация является мета-мета-вычислением.

Итак, первым основным принципом копенгагенского специализатора является наличие мета-мета-уровня, в то время, как в предшествующих специализаторах использовалось только два уровня: базовый уровень и мета-уровень. Соотношение этих трех уровней можно изобразить следующим образом.

Базовый уровень: данные, вычисления над данными, интерпретатор.

Мета-уровень: мета-данные (термы семантического метаязыка), мета-вычисления над мета-данными (частичные вычисления), мета-интерпретатор.

Мета-мета-уровень: мета-мета-данные (значения K и U), мета-мета-вычисления, мета-мета-интерпретатор.

В результате работы первой фазы все параметры функций классифицируются на "известные" K-параметры (устранимые), и "неизвестные" U-параметры (остаточные).

Дальше, на основе информации, полученной в результате KU-анализа, текст исходной программы преобразуется - а н н о т и р у е т с я. Цель аннотирования - превратить глобальную информацию о программе, полученную мета-мета-интерпретацией, в локальную. В процессе аннотирования в программу вставляются дополнительные "руководящие указания", предназначенные для мета-интерпретатора.

Таким образом, вместо того, чтобы ставить над метавычислителем какие-то механизмы, надзирающие за его работой (как это сделано, например, в [ТУР 80], [ТУР 86]), мы заставляем мета-вычислитель самостоятельно выполнять дополнительные указания, расставленные в аннотированной программе. Это - второй принцип, на котором основан копенгагенский специализатор. Этот принцип можно было бы кратко назвать "принцип самоокарауливания".

Понижатель арности московского специализатора в полной

мере следует этим принципам. Эти принципы осуществлены в нем даже более настойчиво и последовательно, чем в копенгагенском, поэтому московский специализатор является в известном смысле более копенгагенским, чем копенгагенский.

2. СЕМАНТИЧЕСКИЙ МЕТА-ЯЗЫК RL

Как указывал еще Й.Футамура [ФУТ 71], в основе системы автоматического построения компиляторов, основанной на самоприменении специализатора, должен лежать "семантический метаязык", удовлетворяющий ряду требований. На этом языке должен быть написан и специализатор, и интерпретаторы, подлежащие преобразованию в компиляторы. В копенгагенском специализаторе в качестве семантического метаязыка использована разновидность чистого Лиспа - язык L.

Автор поставил себе целью построить специализатор, применимый к программам, написанным на Рефале. Поскольку Рефал в свое время был создан в качестве "алгоритмического метаязыка" [ТУР 66,ТУР 68], он, конечно, вполне может служить семантическим метаязыком и действительно используется в качестве такового в системе, описанной в [ТУР 79], [ТУР 80], [ТУР 86].

Рефал содержит средства, позволяющие записывать алгоритмы символьной обработки в простом и наглядном для человека виде (например, сопоставление с образцом). Однако, при построении специализатора, подобного копенгагенскому, эти средства скорее усложняют, чем облегчают работу специализатора. Поэтому, для упрощения специализатора, автор избрал следующий подход.

В качестве семантического метаязыка в московском специализаторе используется язык RL, разработанный автором специально для этой цели. Поэтому Рефал-программы перед

специализацией должны быть предварительно скомпилированы на RL.

Язык RL - это промежуточный язык, который обеспечивает работу с теми же структурами данных, что и Рефал, однако, по сравнению с Рефалом является языком более низкого уровня.

Общая структура RL-программы и управляющие конструкции примерно соответствуют имеющимся в языке Лисп, поэтому обозначение "RL" может быть истолковано как "REFAL-LISP".

RL - менее удобен для человека, чем Рефал, хотя и не до такой степени, чтобы на нем невозможно было программировать непосредственно. Однако, он в некоторых случаях удобнее, чем Рефал, при автоматическом порождении или преобразовании программ.

RL - проще реализуется, чем Рефал, поэтому обратная компиляция сгенерированных RL-программ на Рефал вряд ли целесообразна.

RL-программа представляет собой совокупность определений функций. Функция, которая определяется самой первой, является *начальной* или *входной* функцией программы. Работа программы всегда начинается с вызова этой функции.

Все функции, которые употребляются в RL-программах, делятся на *первичные* и *определяемые*. Первичные функции можно вызывать непосредственно, а определяемые - с помощью первичной функции CALL.

Все определяемые функции имеют фиксированную арность, т.е. определенное количество аргументов (в частности - нулевое). Этим RL отличается от Рефала, в котором все функции унарны (имеют ровно один аргумент). Значением любого аргумента может быть произвольное об'ектное выражение, т.е. произвольная последовательность символов и круглых скобок "(" и

")" , сбалансированная относительно скобок.

Первичные функции имеют фиксированную арность, однако для краткости записи RL-программ принято следующее соглашение: для любой унарной первичной функции можно задать произвольное количество аргументов. В этом случае перед исполнением функции значения этих аргументов конкатенируются в одно выражение, которое и становится фактическим аргументом этой функции.

Первичные функции делятся на следующие категории: вызов определяемой функции - CALL, условное выражение - IF, константа - QUOTE, конструкторы, селекторы, предикаты.

Конструкторы - это функции, которые строят выражения из других выражений. К ним относятся функции BR - надеть скобки, EXPR - тождественная функция. EXPR - унарная функция, поэтому ее можно использовать для конкатенирования нескольких выражений.

Селекторы - это функции, которые позволяют выделять из выражения его части. К ним относятся функции FIRST - взять первый терм выражения, LAST - взять последний терм выражения, BF - взять выражение без его первого терма (сокращение от BUTFIRST), BL - взять выражение без его последнего терма (сокращение от BUTLAST), CONT - взять содержимое скобок.

Предикаты - это функции, которые проверяют истинность условий и вырабатывают символ TRUE, если условие истинно и символ FALSE если оно ложно. К ним относятся функции SYMBOL - проверить, что аргумент является символом, EQUAL - проверить, что два выражения равны.

Синтаксис RL-программ описывается следующим образом (<XXX>* означает повторение конструкции <XXX> произвольное, может быть - нулевое, число раз).

```
<ПРОГРАММА> ::=
      <ОПР-ФУНК> <ОПР-ФУНК>*
<ОПР-ФУНК> ::=
    ( <ИМЯ-ФУНК> ( <ПАРАМ>* ) <RL-ТЕРМ> )
<ИМЯ-ФУНК> ::= <ОБ'ЕКТН-ТЕРМ>
<ПАРАМ> ::= <RL-ПЕРЕМ>
<RL-ПЕРЕМ> ::= <СИМВОЛ>
<RL-ТЕРМ> ::=
    <RL-ПЕРЕМ> | (ABORT)
    | (QUOTE <ОБ'ЕКТН-ВЫРАЖ> )
    | (BR <RL-ВЫРАЖ> )
    | (EXPR <RL-ВЫРАЖ> )
    | (FIRST <RL-ВЫРАЖ> )
    | (BF <RL-ВЫРАЖ> )
    | (LAST <RL-ВЫРАЖ> )
    | (BL <RL-ВЫРАЖ> )
    | (CONT <RL-ВЫРАЖ> )
    | (SYMBOL <RL-ВЫРАЖ> )
    | (EQUAL <RL-ТЕРМ> <RL-ТЕРМ> )
    | (CALL <ИМЯ-ФУНК> <RL-ТЕРМ>* )
    | (IF <RL-ТЕРМ> <RL-ТЕРМ> <RL-ТЕРМ> )
<RL-ВЫРАЖ> ::= <RL-ТЕРМ>*
<ОБ'ЕКТН-ВЫРАЖ> ::= <ОБ'ЕКТН-ТЕРМ>*
<ОБ'ЕКТН-ТЕРМ>::=
    <СИМВОЛ>
    | ( <ОБ'ЕКТН-ВЫРАЖ> )
<СИМВОЛ> ::=
    <БУКВА> |
    <СИМВОЛ> <БУКВА> | <СИМВОЛ> <ЦИФРА> | <СИМВОЛ> -
```

Ниже приведен текст RL-интерпретатора, написанного на RL.

```
( RL-INT (PROGRAM ARGS)
  ( CALL CALL (CONT (FIRST PROGRAM)) ARGS PROGRAM)
)
```

```
(TERM (TERM PARS VALS PROGRAM)
  (IF (SYMBOL TERM)
    (CALL LOOKUPV TERM PARS VALS)
    (CALL TERM-
      (FIRST (CONT TERM))
      (BF (CONT TERM))
      PARS VALS PROGRAM
    )
  )
)

(TERM- (KEY INFO PARS VALS PROGRAM)
  (IF (EQUAL KEY (QUOTE QUOTE))
    INFO
  (IF (EQUAL KEY (QUOTE ABORT))
    (ABORT)
  (IF (EQUAL KEY (QUOTE CALL))
    (CALL CALL
      (CALL LOOKUPF (FIRST INFO) PROGRAM)
      (CALL PARS (BF INFO) PARS VALS PROGRAM)
      PROGRAM
    )
  (IF (EQUAL KEY (QUOTE BR))
    (BR    (CALL EXPR INFO PARS VALS PROGRAM))
  (IF (EQUAL KEY (QUOTE EXPR))
          (CALL EXPR INFO PARS VALS PROGRAM)
  (IF (EQUAL KEY (QUOTE FIRST))
    (FIRST (CALL EXPR INFO PARS VALS PROGRAM))
  (IF (EQUAL KEY (QUOTE BF))
    (BF    (CALL EXPR INFO PARS VALS PROGRAM))
  (IF (EQUAL KEY (QUOTE LAST))
    (LAST  (CALL EXPR INFO PARS VALS PROGRAM))
  (IF (EQUAL KEY (QUOTE BL))
    (BL    (CALL EXPR INFO PARS VALS PROGRAM))
  (IF (EQUAL KEY (QUOTE CONT))
    (CONT  (CALL EXPR INFO PARS VALS PROGRAM))
  (IF (EQUAL KEY (QUOTE SYMBOL))
    (SYMBOL (CALL EXPR INFO PARS VALS PROGRAM))
```

```
  (IF (EQUAL KEY (QUOTE EQUAL))
    (EQUAL
      (CALL TERM (FIRST INFO) PARS VALS PROGRAM)
      (CALL TERM (FIRST (BF INFO)) PARS VALS PROGRAM)
    )
  (IF (EQUAL KEY (QUOTE IF))
    (IF (CALL TERM (FIRST INFO) PARS VALS PROGRAM)
      (CALL TERM (FIRST (BF INFO)) PARS VALS PROGRAM)
      (CALL TERM (FIRST (BF (BF INFO))) PARS VALS PROGRAM)
    )
  (ABORT)
  ))))))))))))
)

(CALL (FNDEF VALS PROGRAM)
  (CALLR TERM
    (LAST FNDEF)
    (CONT (FIRST (BF FNDEF)))
    VALS PROGRAM
  )
)

(PARS (TERMS PARS VALS PROGRAM)
  (IF (EQUAL TERMS (QUOTE))
    (QUOTE)
    (EXPR
      (BR (CALL TERM (FIRST TERMS) PARS VALS PROGRAM) )
      (CALL PARS (BF TERMS) PARS VALS PROGRAM)
    )
  )
)

(EXPR (TERMS PARS VALS PROGRAM)
  (IF (EQUAL TERMS (QUOTE))
    (QUOTE)
    (EXPR
      (CALL TERM (FIRST TERMS) PARS VALS PROGRAM)
      (CALL EXPR (BF TERMS) PARS VALS PROGRAM)
```

```
    )
  )
)

(LOOKUPV (VAR PARS VALS)
  (IF (EQUAL PARS (QUOTE))
    (ABORT)
  (IF (EQUAL VAR (FIRST PARS))
    (CONT (FIRST VALS))
    (CALL LOOKUPV VAR (BF PARS) (BF VALS))
  ))
)

(LOOKUPF (FNAME PROGRAM)
  (IF (EQUAL FNAME (FIRST (CONT (FIRST PROGRAM))))
    (CONT (FIRST PROGRAM))
    (CALL LOOKUPF FNAME (BF PROGRAM))
  )
)
```

3. ОБЩАЯ СТРУКТУРА СПЕЦИАЛИЗАТОРА

При решении всякой задачи полезно не терять из виду различие между целью, которая должна быть достигнута, и методами, которые приходится применять для ее достижения. В контексте данной работы специализация программ является целью, а частичные вычисления – одним из методов ее достижения. Однако, не из чего не следует, что специализация программ может достигаться только частичными вычислениями. Полезны и другие средства (в числе которых находятся и традиционные методы оптимизации).

Московский специализатор, в действительности, состоит из двух основных частей: понижателя арности и повышателя арности.

Понижатель арности в целом соответствует копенгагенскому специализатору. Он основан на принципе частичных, или, как мы предпочитаем их называть, мета-вычислений. Особенностью этого метода специализации является то, что функции, входящие в остаточную программу, имеют не больше аргументов, чем те функции исходной программы, из которых они были порождены. Это и оправдывает название "понижатель арности".

Программа, построенная понижателем арности, затем поступает на вход повышателя арности, который основан на более или менее традиционных методах оптимизации.

Обращение к специализатору имеет следующий вид:

```
(CALL SPEC PROG PARS-CL K-VALS U-TYPES)
```

где аргументы имеют следующий смысл:

PROG - исходная RL-программа, подлежащая специализации;

PARS-CL - последовательность символов "K" и "U", которая описывает, какие из параметров программы PROG будут известны (K), а какие - неизвестны (U), во время специализации;

K-VALS - значения K-параметров;

U-TYPES - типы U-параметров.

Функция SPEC вызывает две другие функции: понижатель арности DECR-AR и повышатель арности INCR-AR.

```
(SPEC (PROG PARS-CL K-VALS U-TYPES)
  (CALL INCR-AR
    (CALL DECR-AR PPOG PARS-CL K-VALS)
    U-TYPES
  )
)
```

4. ПОВЫШАТЕЛЬ АРНОСТИ

Работа повышателя арности распадается на три этапа:

- выяснение типов параметров и результатов функций;
- расщепление параметров функций на основе информации о типах параметров;
- локальная оптимизация на основе информации о типах параметров и результатов функций.

Так, например, если в результате анализа типов выясняется, что значение переменной X обязательно имеет вид

```
( E1 ) E2 T3
```

где E1, E2 – произвольные об'ектные выражения, а T3 – произвольный об'ектный терм, параметр X расщепляется на три параметра X1, X2, X3, в которых хранятся выражения E1, E2, T3. При этом все вхождения параметра X заменяются на

```
(EXPR (BR X1) X2 X3)
```

Таким образом, работа повышателя арности основана на достаточно хорошо известных принципах. Поэтому в данной работе он не будет описываться более подробно. Тем не менее, следует обратить внимание на следующие обстоятельства:

- повышатель арности является естественным дополнением к понижателю арности, ибо их действия хорошо дополняют друг друга;
- сравнительно простыми средствами повышатель арности заметно улучшает качество программ, порождаемых специализатором;

- повышатель арности работает полностью автоматически, поэтому отпадает необходимость в дополнительном ручном аннотировании программ [ДСЗ 85Б], [СЕС 86] с целью повышения арности.

В дальнейшем, при обсуждении различий между московским и копенгагенским специализаторами под московским специализатором будет пониматься главным образом понижатель арности, ибо автоматический повышатель арности в копенгагенском специализаторе отсутствует.

5. ПОНИЖАТЕЛЬ АРНОСТИ

Понижатель арности уничтожает К-параметры программы. Его работа протекает в три этапа.

Сначала работает мета-мета-интерпретатор, который, исходя из описания классов входных параметров программы, выясняет для каждой функции, какие из ее параметров являются К-параметрами, а какие U-параметрами.

Кроме того, все функции исходной программы разделяются на две категории: К-функции, которые в процессе мета-интерпретации будут всегда выдавать К-значения и U-функции, которые в процессе мета-интерпретации всегда будут выдавать U-значения.

> В этом - одно из отличий московского специализатора от копенгагенского, ибо в копенгагенском специализаторе классифицируются только параметры, но не сами функции.

Затем выясняется, какие из U-функций являются о с т а т о ч н ы м и, т.е. вызываются с помощью CALLR хотя бы из одной U-функции.

На основании полученной информации описания U-функций аннотируются, т.е. преобразуются некоторым образом, чтобы облегчить работу мета-интерпретатора. Как именно делается аннотирование будет описано ниже.

Результатом работы мета-мета-интерпретатора является выражение, которое расчленяется на три компоненты и передается в виде трех отдельных аргументов мета-интерпретатору. Смысл этих аргументов будет описан ниже.

Мета-интерпретатор производит построение специализированной версии исходной программы, используя фактические значения K-параметров K-VALS.

И наконец, результат специализации передается функции RENAME-FUNCS, которая придумывает для функций, входящих в специализированную программу, более короткие имена.

Таким образом, понижатель арности описывается следующим образом:

```
(DECR-AR (PROG PARS-CL K-VALS)
  (CALL DECR-AR-
    (CALL MM-INT PROG PARS-CL)
    K-VALS
  )
)

(DECR-AR- (ANN-PROG K-VALS)
  (CALL RENAME-FUNCS
    (CALL M-INT
      (CONT (FIRST ANN-PROG))
      (CONT (FIRST (BF ANN-PROG)))
      (CONT (FIRST (BF (BF ANN-PROG))))
      K-VALS
    )
  )
)
```

6. СТРУКТУРА АННОТИРОВАННОЙ ПРОГРАММЫ

Аннотированная программа имеет следующий вид:

(R-FUNCS) (K-PROG) (U-PROG)

где выражения R-FUNCS, K-PROG и U-PROG имеют следующий смысл:

R-FUNCS – имена остаточных функций, т.е. функций, частные случаи которых могут попасть в остаточную программу;

K-PROG – K-программа, т.е. определения функций, все параметры которых принадлежат классу K, и значение которых всегда принадлежит классу K;

U-PROG – U-программа, т.е. определения функций, имеющих хотя бы один параметр класса U и всегда выдающих значения класса U.

Определения функций, входящие в K-программу, берутся в неизменном виде из исходной программы.

Определения функций, входящие в U-программу, представляют собой определения соответствующих функций, взятые из исходной программы и преобразованные следующим образом.

Пусть исходное определение функции имело вид

(F (X1 X2 ... XL) T)

тогда это определение преобразуется к виду

(F (K1 K2 ... KM) (U1 U2 ... UN) AT)

где K-i – это те из X-i, которые принадлежат классу K, а U-i

– это те из X–İ, которые принадлежат классу U (при этом M+N=L). AT – это RL–терм, который получается в результате аннотирования терма T следующим образом.

Будем называть K–термами такие RL–термы, которые не содержат ни U–параметров, ни вызовов U–функций. RL–термы, которые не являются K–термами, мы будем назвать U–термами.

Пусть T – некоторый RL–терм. Тогда через AT будем обозначать результат аннотирования терма T.

Для всякого RL–терма T, результат его аннотирования AT получается согласно следующим правилам (приоритет применения которых определяется порядком, в котором они выписаны).

Если T = (QUOTE C), то AT = (QUOTE C).

Если T = (ABORT), то AT = (ABORT).

Если T является K–термом, то AT = (META T).

Если T – переменная, то AT = T.

Если T = (CALL F T1 T2 ... TL), то AT = (CALL F (K1 K2 ... KM) (AU1 AU2 ... AUN)), где M+N=L, K1 K2 ... KM – это перечень всех T–İ, находящихся на местах, соответствующих K–параметрам функции F, а AU1 AU2 ... AUN – это перечень термов, получающихся аннотированием тех T–İ, которые находятся на местах, соответствующих U–параметрам функции F.

Если T = (CALLR F T1 T2 ... TL), то AT = (CALLR F (K1 K2 ... KM) (AU1 AU2 ... AUN)), где M+N=L, K1 K2 ... KM – это перечень всех T–İ, находящихся на местах, соответствующих K–параметрам функции F, а AU1 AU2 ... AUN – это перечень термов, получающихся аннотированием тех T–I, которые находятся на местах, соответствующих U–параметрам функции F.

Если T = (IF TØ T1 T2), и TØ – K–терм, то AT = (IF–E TØ

AT1 AT2).

Если T = (IF TØ T1 T2), и TØ – U-терм, то AT = (IF-R ATØ AT1 AT2).

Если T = (P T1 T2 ... TN), где P – имя первичной функции BR, EXPR, FIRST, BF, LAST, BL, CONT, SYMBOL или EQUAL, то AT = (P AT1 AT2 ... ATN).

Например, рассмотрим функцию:

```
(ZIPPER (X Y)
  (IF (EQUAL X (QUOTE))
    Y
  (IF (EQUAL Y (QUOTE))
    X
    (EXPR
      (FIRST X) (FIRST Y)
      (CALL ZIPPER (BF X) (BF Y))
    )
  ))
)
```

Если X является K-параметром, а Y – U-параметром, результатом аннотирования функции ZIPPER будет

```
(ZIPPER (X)(Y)
  (IF-E (EQUAL X (QUOTE))
    Y
  (IF-R (EQUAL Y (QUOTE))
    (META X)
    (EXPR
      (META (FIRST X)) (FIRST Y)
      (CALL ZIPPER ((BF X)) ((BF Y)) )
    )
  ))
)
```

7. МЕТА-ИНТЕРПРЕТАТОР

Обращение к мета-интерпретатору имеет следующий вид

```
(CALL M-INT R-FUNCS K-PROG U-PROG K-VALS)
```

Первые три параметра содержат аннотированную программу, подлежащую специализации, а параметр K-VALS - значения K-параметров этой программы. Результатом работы мета-интерпретатора является специализированная программа, состоящая из функций, которые являются специализированными версиями функций, входящих в U-PROG. А именно, если определение U-функции имеет вид

```
(F (K1 ... KM) (U1 ... UN) T)
```

то из нее могут порождаться описания функций вида

```
((F (C1) ... (CM)) (U1 ... UN) T')
```

где C1,..., CM - константы, являющиеся значениями K-параметров K1,..., KM, а T' - результат мета-вычисления терма T при условии, что значениями K-параметров являются C1,..., CM, а значениями U-параметров являются сами эти параметры U1,..., UN. (F (C1) ... (CM)) - имя сгенерированной функции.

Таким образом, основная задача мета-интерпретатора, это мета-вычисление RL-термов. Результатом такого вычисления являются U-значения, т.е. RL-термы.

Рассмотрим, например, определенную выше функцию ZIPPER. Она имеет K-параметр X и U-параметр Y. Предположим, что метавычислителю требуется вычислить обращение к функции ZIPPER, при условии, что параметры X и Y приняли следующие значения:

```
X = "ONE TWO"; Y = "VAR"
```

(здесь и далее мы заключаем значения переменных в кавычки, чтобы они не путались с именами переменных). Тогда результатом вызова функции ZIPPER будет RL-терм

```
(IF (EQUAL VAR (QUOTE))
  (QUOTE ONE TWO)
  (EXPR
    (QUOTE ONE) (FIRST VAR)
    (IF (EQUAL (BF VAR) (QUOTE))
      (QUOTE TWO)
      (EXPR
        (QUOTE TWO) (FIRST (BF VAR))
        (BF (BF VAR))
      )
    )
  )
)
```

При метавычислении тела функции ZIPPER, метавычислитель два раза выполняет рекурсивный вызов функции ZIPPER. При этом параметры X и Y принимают после первого вызова значения

$$X = \text{"TWO"}; \quad Y = \text{"(BF VAR)"}$$

а при втором вызове - значения

$$X = \text{""}; \quad Y = \text{"(BF (BF VAR))"}.$$

Из сказанного очевидно, что самая сложная из операций, выполняемых мета-интерпретатором, - это мета-вычисление RL-термов. Именно в этом пункте и заключается основное отличие московского специализатора от копенгагенского.

Основной принцип, на котором построен московский специализатор состоит в том, что мета-интерпретатор, в отличие от обычного интерпретатора, должен работать с двумя классами значений К-значениями и О-значениями. Эти значения имеют принципиально разную природу, поскольку принадлежит к

различным уровням: K-значения - к базовому уровню, а U-значения - к мета-уровню. По отношению к K-значениям U-значения являются мета-значениями, ибо они представляют собой RL-термы, которые в дальнейшем, во время исполнения специализированной программы, будут вырабатывать K-значения.

Таким образом, операции над K-значениями и U-значениями должны выполняться совершенно по-разному. Например, применение операции BF к K-значению "ONE TWO THREE" дает K-значение "TWO THREE", а применение операции BF к U-значению "(EXPR X1 X2)" дает U-значение "(BF (EXPR X1 X2))".

Из сказанного следует, что мета-интерпретатор может использовать совершенно различное представление для K-значений и U-значений. А именно, K-значения естественно хранить и обрабатывать прямо в "натуральном" виде, т.е. в том же виде, в котором их хранил бы обычный RL-интерпретатор, а U-значения естественно хранить в виде RL-термов. При этом операции над K-значениями выполняются "реально", т.е. как в RL-интерпретаторе, а операции над U-значениями выполняются "номинально", т.е. в мета-смысле (что сводится к навешиванию на RL-терм снаружи символа соответствующей операции).

Таким образом, первый основной принцип, на котором построен московский специализатор, заключается не в том, чтобы смешивать K- и U-вычисления, а в том чтобы их тщательно р а з д е л я т ь. Этот принцип можно было бы назвать принципом р а з д е л ь н ы х (UNMIXED) вычислений.

Второй основной принцип заключается в том, что всякое вычисление над K- и U-значениями можно и следует доводить до конца. Поскольку и K- и U-значения специализатору полностью известны, он имеет полную возможность выполнить над ними все операции (хотя способ выполнения для K- и U-значений разный). Таким образом, нет особых оснований говорить о частичных вычислениях. Этот принцип можно было бы назвать принципом

п о л н ы х (NON-PARTIAL) вычислений.

Сказанное выше об'ясняет, почему автор предпочитает термин "метавычисления" термину частичные вычисления когда речь идет о московском специализаторе.

Общая структура мета-интерпретатора является следствием изложенных выше принципов. Основу мета-интерпретатора составляют две функции: EVAL-TERM и SPEC-TERM. Функция EVAL-TERM вычисляет К-значение К-терма при условии, что заданы значения К-параметров и К-программа. Функция SPEC-TERM вычисляет U-значение U-терма при условии, что заданы значения К-параметров, U-параметров, К-программа и U-программа.

Таким образом, мета-интерпретатор содержит К-интерпретатор, который является обычным RL-интерпретатором, и U-интерпретатор, который является мета-интерпретатором в собственном смысле слова.

К-интерпретатор, будучи вызван, уже никогда не обращается к U-интерпретатору. U-интерпретатор, напротив, вызывает К-интерпретатор в тех случаях, когда подтермами вычисляемого U-терма являются К-термы. А именно, если ему встречается одна из конструкций

```
(META T)
(IF-E T T1 T2)
(CALL  F (K1 ... KM) (U1 ... UN) )
(CALLR F (K1 ... KM) (U1 ... UN) )
```

Какой терм каким интерпретатором вычислять всегда ясно по контексту. Вследствие этого нет необходимости аннотировать первичные функции, как это делается в копенгагенском специализаторе (например, вместо одной функции FIRST иметь в аннотированной программе две функции FIRST-E и FIRST-R). Исключением является только конструкция IF.

8. РЕЗУЛЬТАТЫ САМОПРИМЕНЕНИЯ СПЕЦИАЛИЗАТОРА

Особенностью копенгагенского и московского специализаторов является то, что для получения компиляторов из интерпретаторов достаточно вместо выражения SPEC(SPEC,INT) вычислять SPEC(M-INT,ANN-INT), где M-INT - мета-интерпретатор, а ANN-INT - результат аннотирования интерпретатора INT. Аналогично, для порождения генератора компиляторов достаточно вычислить SPEC(M-INT,ANN-M-INT), где ANN-M-INT - результат аннотирования мета-интерпретатора M-INT в предположении, что его параметры принадлежат следующим классам: R-FUNCS - K, K-PROG - K, U-PROG - K, K-VALS - U. Это обусловлено тем, что мета-интерпретатор - это единственная часть специализатора, которая получает в качестве одного из аргументов K-VALS (значения входных K-параметров) [ДСЗ 85А], [ДСЗ 85Б], [СЕС 86].

Автором была осуществлена генерация компиляторов из нескольких интерпретаторов. А именно, были рассмотрены интерпретатор языка XY (по существу совпадающего с императивным языком, описанным в [ДСЗ 85Б]), интерпретатор конечных автоматов, RL-интерпретатор, интерпретатор ограниченного Рефала [ТУР 86].

Оказалось, что все порожденные компиляторы имеют вполне естественную, с человеческой точки зрения, структуру и легко читаются. Порожденный RL-компилятор, как и следовало ожидать, оказался скорее оптимизатором RL-программ, чем компилятором, ибо его входным и выходным языком является RL.

Далее, специализацией мета-интерпретатора на мета-интерпретатор был вычислен генератор компиляторов. Его текст, как и следовало ожидать, содержал бессмысленные (с точки зрения человека) имена функций и параметров функций (например SPEC-TERM-1, SPEC-TERM-2,..., SPEC-TERM-45). Однако, после того, как автор вручную заменил эти имена на более содержательные, оказалось, что текст генератора компиляторов вполне

поддается прочтению.

Изучение текста генератора компиляторов позволило разобраться в принципах его работы. В результате стало ясно, каким способом получились из интерпретаторов все автоматически построенные компиляторы. Более того. Поскольку сам генератор компиляторов получен из мета-интерпретатора в соответствии с этими же принципами, прояснилась и связь между мета-интерпретатором и генератором компиляторов.

9. КАК ИНТЕРПРЕТАТОР ПРЕВРАЩАЕТСЯ В КОМПИЛЯТОР

Все автоматически построенные компиляторы состоят из двух частей: административной части, которая организует работу компилятора в целом, и генерирующей части, которая осуществляет непосредственное порождение результирующей программы. Подобным образом устроены и компиляторы, порожденные копенгагенским специализатором [ДСЗ 85Б], [СЕС 86].

Административная часть имеет регулярную структуру, которая почти не зависит от исходного интерпретатора. Она представляет собой слегка специализированную версию административной части мета-интерпретатора.

Генерирующая часть совершенно различна для разных компиляторов и полностью определяется структурой исходных интерпретаторов.

Аннотированный интерпретатор, как говорилось выше, имеет вид

(R-FUNCS) (K-PROG) (U-PROG)

Функции, входящие в K-PROG, переносятся в компилятор без изменения (с точностью до переименования имен функций и

переменных и с локальными оптимизациями).

Каждой функции, имя которой входит в R-FUNCS, в компиляторе соответствует некоторая функция, которая получается из соответствующей функции, входящей в U-PROG, по описанным ниже правилам (с точностью до переименования функций и переменных и с локальными оптимизациями).

Функции, входящие в U-PROG, будем называть "интерпретирующими", а порожденные из них функции, входящие в компилятор, будем называть "компилирующими". Кроме того, для всякого RL-терма T, входящего в описание интерпретирующей функции, будем обозначать через CT RL-терм, который ему соответствует в описании компилирующей функции.

Если определение интерпретирующей функции F имеет вид

(F (K1 ... KM) (U1 ... UN) T)

то ей ставится в соответствие компилирующая функция вида

(U-F (K1 ... KM U1 ... UN) CT)

Для всякого RL-терма T, терм CT получается из T согласно следующим правилам (приоритет применения которых соответствует порядку, в котором они выписаны).

Если T = (QUOTE C), то CT = (QUOTE (QUOTE C)).

Если T = (ABORT), то CT = (QUOTE (ABORT)).

Если T = (META T1), то CT = (BR (QUOTE QUOTE) T1).

Если T - переменная, то CT = T.

Если T = (CALL F (K1 ... KM) (U1 ... UN)), то CT = (CALL U-F K1 ... KM CU1 ... CUN).

Если T = (CALLR F (K1 ... KM) (U1 ... UN)), то CT = (BR (QUOTE CALL) (BR (QUOTE F) (BR K1) ... (BR KM)) CU1 ... CUN).

Если T = (IF-E TØ T1 T2), то CT = (IF TØ CT1 CT2).

Если T = (IF-R TØ T1 T2), то CT = (BR (QUOTE IF) CTØ CT1 CT2).

Если T = (P T1 ... TN), где P - имя первичной функции BR, EXPR, FIRST, BF, LAST, BL, CONT, SYMBOL или EQUAL, то CT = (BR (QUOTE P) CT1 ... CTN).

Например, из рассмотренной ранее функции ZIPPER получается компилирующая функция

```
(U-ZIPPER (X Y)
  (IF (EQUAL X (QUOTE))
    Y
  (BR (QUOTE IF)
    (BR (QUOTE EQUAL) Y (QUOTE (QUOTE)))
    (BR (QUOTE QUOTE) X)
    (BR (QUOTE EXPR)
      (BR (QUOTE QUOTE) (FIRST X))
      (BR (QUOTE FIRST) Y)
      (CALL U-ZIPPER (BF X) (BR (QUOTE BF) Y)
    )
  ))
)
```

Описанные правила порождения компилирующих функций из интерпретирующих довольно естественны. Не исключено, что подобные правила использовались еще в написанном вручную генераторе компиляторов, описанном в [БЕК 76]. Интересным, однако, является то, что в нашем случае эти правила были автоматически "изобретены" компьютером в процессе специализации специализатора.

Не следует забывать, что описанные правила построения компилирующих функций описывают результат работы генератора компиляторов как единого целого, включающего автоматический повышатель арности. Если бы повышатель арности был исключен из генератора компиляторов, получались бы компилирующие функции имеющие ровно два параметра: K-VALS и U-VALS. K-VALS содержал бы K-значения, а U-VALS - U-значения.

З А К Л Ю Ч Е Н И Е

Основной особенностью московского специализатора по сравнению с копенгагенским является более жесткое и статическое разграничение K-значений и U-значений. Это достигается следующими мерами.

- Помимо разделения на K-параметры и U-параметры введено разделение на K-функции и U-функции.

- Введен новый способ аннотирования программ, при котором программа делится на K-программу и U-программу, и нет необходимости вводить для каждой первичной функции P два имени: P-E и P-R.

- Новый способ аннотирования позволил разделить мета-интерпретатор на K-интерпретатор и U-интерпретатор. K-интерпретатор является обычным RL-интерпретатором и работает только с K-значениями.

- Разделение на K-интерпретатор и U-интерпретатор позволило использовать различные представления для K-значений и U-значений. K-значения являются об'ектными выражениями, а U-значения - RL-термами.

Улучшение структуры остаточных программ достигнуто главным образом, за счет автоматического повышения арности и

использования различного представления для К- и U-значений.

В копенгагенском специализаторе К-значениями являются не сами константы, а их изображения средствами семантического метаязыка. Т.е. вместо самой константы "С" в качестве К-значения используется "(QUOTE C)". Это приводит к следующим последствиям. Пусть, например, в исходном интерпретаторе имеется конструкция "(FIRST X)", где X - К-переменная. Тогда при использовании единого представления для К- и U-значений в сгенерированном компиляторе появляется конструкция

(BR (QUOTE QUOTE) (FIRST (BF (CONT X))))

между тем, московский специализатор создаст в компиляторе только конструкцию "(FIRST X)".

С другой стороны, разделение мета-интерпретатора на К-интерпретатор и U-интерпретатор позволило отказаться от попыток сразу же выполнять локальную оптимизацию порождаемых U-значений. Ведь эта оптимизация все равно будет выполняться в повышателе арности. При этом качество оптимизации будет выше, ибо будет возможность использовать глобальную информацию о программе, полученную в результате анализа типов параметров и результатов функций. Таким образом, конструкция "(FIRST X)", где X - U-переменная, в сгенерированном компиляторе переходит в "(BR (QUOTE FIRST) X)".

Из сказанного очевидно, что использование различного представления для К- и U-значений приводит к прояснению структуры как генератора компиляторов, так и порождаемых им компиляторов.

Б Л А Г О Д А Р Н О С Т Ь

Автор выражает благодарность С.М.Абрамову, Ан.В.Климову, Ар.В.Климову, Н.В.Кондратьеву, В.Л.Кистлерову, А.Ю.Романенко и другим членам рабочей группы по языку Рефал при Комиссии по языкам и системам программирования ГКНТ СССР. В процессе выполнения данной работы ее результаты многократно обсуждались на заседаниях рабочей группы, что служило для автора постоянным стимулом к работе и способствовало прояснению и уточнению излагаемых в работе концепций.

Автор также пользуется возможностью выразить благодарность Вс.С.Штаркману за создание деловой, творческой атмосферы, способствовавшей успешному выполнению данной работы.

Л И Т Е Р А Т У Р А

[БЕК 76]
L.BECKMAN, A.HARALDSON, O.OSKARSSON, E.SANDEWALL. A PARTIAL EVALUATOR, AND ITS USE AS PROGRAMMING TOOL. - ARTIFICIAL INTELLIGENCE, VOL.7, NO.4, 1976, PP.319–357.

[ДИК 71]
J.DIXON. THE SPECIALIZER, A METHOD OF AUTOMATICALLY WRITING COMPUTER PROGRAMS. - DIVISION OF COMPUTER RESEARCH AND TECHNOLOGY, NATIONAL INST. OF HEALTH, BETHENDA, MARYLAND, 1971.

[ДСЗ 85А]
N.D.JONES, P.SESTOFT, H.SONDERGAARD. AN EXPERIMENT IN PARTIAL EVALUATION: THE GENERATION OF A COMPILER GENERATOR. - SIGPLAN NOTICES, VOL.20, NO.8, 1985, PP.82–87.

[ДСЗ 85Б]
N.D.JONES, P.SESTOFT, H.SONDERGAARD. AN EXPERIMENT IN PARTIAL EVALUATION: THE GENERATION OF A COMPILER GENERATOR. - IN PROC. 1ST INTL. CONF. ON REWRITING TECHNIQUES AND APPLICATIONS, DIJON, FRANCE, 1985. SPRINGER LNCS 202 (1985), PP.124–140.

[ЛОМ 67]
L.A.LOMBARDI. INCREMENTAL COMPUTATION. - ADVANCES IN COMPUTERS, 8, ACADEMIC PRESS, NEW YORK, 1967.

[СЕС 86]
P.SESTOFT. THE STRUCTURE OF A SELF-APPLICABLE PARTIAL EVALUATOR. - IN H.GANZINGER AND N.D.JONES (EDS.): PROGRAMS AS DATA OBJECTS, COPENHAGEN, DENMARK, 1985. LECTURE NOTES IN COMPUTER SCIENCE 217 (1986), PP. 236–256. SPRINGER-VERLAG.

[ТНТ 82]
V.F.TURCHIN, R.N.NIRENBERG, D.V.TURCHIN. EXPERIMENTS WITH THE SUPERCOMPILER. – CONFERENCE RECORD OF THE ACM SYMPOSIUM ON LISP AND FUNCTIONAL PROGRAMMING, 1982, PP.47–55.

[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 68]
В.Ф.Турчин. Метаалгоритмический язык. Кибернетика, N 4, 1968, с.45–54.

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

[ТУР 79]
V.F.TURCHIN. SUPERCOMPILER SYSTEM BASED ON THE LANGUAGE REFAL. – SIGPLAN NOTICES, VOL.14, No.2, 1979, pp.46–54.

[ТУР 80]
V.F.TURCHIN. SEMANTIC DEFINITIONS IN REFAL AND THE AUTOMATIC PRODUCTION OF COMPILERS, IN LNCS 94: SEMANTICS DIRECTED COMPILER GENERATION (N.D.JONES, ED.), PP.441–474, SPRINGER 1980.

[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.

[ФУТ 71]

Y.FUTTAMURA. PARTIAL EVALUATION OF COMPUTATION PROCESS – AN APPROACH TO A COMPILER COMPILER. – SYSTEMS COMPUTERS CONTROLS, VOL.2, NO.5, 1971, PP.45–50.

[ЧНЛ 73]

CH.CHANG, R.LEE. SYMBOLIC LOGIC AND MECHANICAL THEOREM PROVING. – ACADEMIC PRESS, 1973. (Имеется русский перевод: Ч.Чень, Р.Ли. Математическая логика и автоматическое доказательство теорем: Пер. с англ./Под ред. С.Ю.Маслова. – М.: Наука, 1983. – 360 с.)

Романенко Сергей Анатольевич." Генератор компиляторов, порожденный самоприменением специализатора, может иметь ясную и естественную структуру."
Редактор В.С. Штаркман. Корректор А.В. Климов.

Подписано к печати 21.01.87г. № Т- 04808. Заказ № 62.
Формат бумаги 60X90 1/16. Тираж 220 экз.
Объём 1,6 уч.-изд.л. Цена 13 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 13 коп.