244-127 23.5.77

О сущности трансляции

А.П.Ершов
Вычислительный центр СО АН СССР
Новосибирск 630090

Трансляция обычно рассматривается состоящей из трех фаз: декомпозиции, оптимизации и генерации. В отличие от хорошо разработанной синтаксической стороны трансляции, трактовка остальных двух фаз носит зачастую весьма произвольный характер. Автор пытается вскрыть сущность генерации и значительной части оптимизации, используя понятие смешанного вычисления, вводимого как фундаментальное понятие, присущее алгоритмическим языкам. Рассматриваются вопросы получения объектного кода непосредственно из интерпретационной семантики языка, некоторые важные виды направленной оптимизации, а также систематическое получение трансляционной семантики из интерпретационной. Дается обзор других работ, имеющих отношение к рассматриваемому вопросу.

I. СМЕШАННЫЕ ВЫЧИСЛЕНИЯ

I.I. Основная идея. Понятие смешанного вычисления опирается на один общеизвестный математический факт.

Пусть нам дана функция двух переменных $f(x,y)$. Частичным связыванием аргумента называется подстановка в функцию значения одного из аргументов, скажем $x=a$:

$$f(x,y)\Big|_{x=a} \Longrightarrow \varphi(y) .$$

Из этого изображения следует, что частичное связывание аргумента - это двойной процесс. С одной стороны - это некоторое вычисление,

241-12 р

которое становится возможным в связи с заданием значений аргументов; с другой стороны - это оператор, преобразующий функцию $z(x,y)$ в $\varphi(y)$.

Основная часть изложения будет вестись на примере очень простого языка МИЛАН (MIni-LANguage), заимствованного из [1]. Он содержит целые переменные и константы, знаки операций и предикаты и образованные из них функциональные и предикатные термы. Базисными операторами являются операторы присваивания, ввода и вывода, составными операторами - условные операторы и циклы.

Ниже следует входной синтаксис языка МИЛАН:

<программа> ::= нач <серия> кон

<серия> ::= <оп> | <оп>; <серия>

<опер> ::= <присв> | <чит> | <выв> | <усл> | <цикл>

<чит> ::= чит <перлист>

<зап> ::= зап <перлист>

<перлист> ::= <пер> | <пер>, <перлист>

<присв> ::= <пер> ::= <выр>

<выр> ::= <втор> | <выр> <слож> <втор>

<слож> ::= + | -

<втор> ::= <элем> | <втор> <умн> <элем>

<умн> ::= × | ÷

<элем> ::= <пер> | <конст> | (<выр>)

<пер> ::= <бук> | <пер> <бук> | <пер> <циф>

<конст> ::= <num> | -<num>

<num> ::= <циф> | <num> <циф>

<бук> ::= a | b | c | ... | z

<циф> ::= 0 | 1 | 2 | ... | 9

241-129

<если> ::= <если-то> <иначе> | <если-то> <иначе> <серия> все

<если-то> ::= если <выр> <отн> <выр> то <серия>

<отн> ::= =|>|<|≤|≥|≠

<цикл> ::= пока <выр> <отн> <выр> цк <серия> кц

Рассмотрим в языке МИЛАН фрагмент программы, вычисляющей $y=x^n$ с использованием двоичного разложения показателя (надеемся, что читатель простит вольность употребления еще одного примитива нечет (n), распознающего нечетность n):

Программа 1

y:=1; пока n >0 цк если нечет (n) то y:=y × x; n := n −1 все;
n := n ÷ 2 ; x:=x × x кц .

Свяжем в функции $y=x^n$ аргумент n , положив n=5 . Очень легко написать программу, вычисляющую $y=x^5$:

Программа 2

y:=x; x:=x × x; x:=x × x; y:=y × x .

Нас, однако, интересует другой вопрос: нельзя ли получить подобную программу из общей программы, применяя к последней какую-то систематическую процедуру? Как будет видно из обсуждения, этот вопрос интересовал разных авторов. Мы предложим еще один подход к этому вопросу.

Рассмотрим так называемую операционную историю, или прокрутку, программы 1 для n =5 и какого-нибудь x , скажем x=3 . Это будет последовательность всех выполнений операторов и вычислений предикатов:

Программа 3

y:=1; (n >0)⁺; нечет(n)⁺; y:=y × x; n := n −1; n:=n÷2 ;x:=x×x;
(n >0)⁺; нечет(n)⁻; n :=n÷2 ;x:=x×x;
(n >0)⁺; нечет(n)⁺; y:=y × x; n := n −1; n :=n÷2 ;x:=x×x;
(n >0)⁻ .

241-130

нам будет удобно считать программный текст (программу 1) только генератором операционной истории, полагая, что "фактическое" вычисление производится операторами линейной программы (программа 3), порождающей истории.

Вернемся теперь к проблеме получения программы для $y=x^5$. Попробуем выполнить программу 3 при незаданном x. Очевидно, что все операторы и предикаты, как-то связанные с x, не могут выполняться; все эти "задержанные" операторы будем складывать в порядке их обнаружения где-то в другом месте. Сразу же, чтобы сделать изложение более однообразным, будем считать, что выполняются все операторы, при этом незадержанные операторы выполняются обычным образом, а задержанные операторы выполняются "литерально", т.е. выдают в качестве результата самих себя. Литеральное значение задержанного оператора не обязательно тождественно ему самому: если оператор содержит терм, значение которого может быть вычислено, то терм заменяется изображением своего текущего значения (разгрузка оператора). Последовательно накапливаемые литеральные значения операторов образуют некоторый текст, который по окончанию вычисления будет называться остаточной программой. Применив эти интуитивные соображения к программе 3, мы осуществим некоторое частичное вычисление:

Частичная история

y:=1;	$(n>0)^+$;	нечет $(n)^+$;	$n := n-1$;	$n := n+2$;
	$(n>0)^+$;	нечет $(n)^-$		$n := n+2$;
	$(n>0)^+$;	нечет $(n)^+$;	$n := n-1$;	$n := n+2$;
	$(n>0)^-$			

и получим остаточную программу

241-139

Программа 4

y:=T^x; x:=x^x; x:=x^x; y:=y^x; x:=x^x ,

очень близкую к программе 2.

Первый оператор в программе 4 отличается от своего прообраза заменой терма y на его значение. Заметим, что наше решение "заморозить" x в программе 1 было произвольным, однако все остальные операторы, попавшие в остаточную программу по индукции, были задержаны "вынужденно", т.к. их термы зависели от замороженной переменной x . Наконец,мы видим, что так определенное выполнение программы носит действительно дуальный характер: это, с одной стороны, некоторое частичное вычисление и, с другой стороны, отображение исходной программы на другую остаточную программу. Это и есть то, что мы называем смешанным вычислением.

Дадим сперва самое общее определение смешанного вычисления.

Пусть в языке L определен алгорити вычисления V , который для любой программы P и начального состояния памяти X иногда приводит к некоторому заключительному состоянию памяти Y =V(P,X).

Смешанным вычислением в языке L называется любой алгоритм M , который для любой программы P и начального состояния памяти X иногда приводит к некоторому промежуточному состоянию памяти Y' =M_C(P,X) и к некоторой остаточной программе P'=M_G(P,X) . Смешанное вычисление M корректно в L , если для любых P и X справедливо следующее функциональное равенство

$$V(P,X) = V(M_G(P,X),\ M_C(P,X))$$

(принцип частичного вычисления).

I.2. Смешанные вычисления в языке МИЛАН. Обычное вычисление в языке МИЛАН - это последовательность вычислений термов и выполнения операторов. В смешанном вычислении каждый из этих вычисли-

241-132

тельных актов может быть объявлен произвольно задержанным. Причина произвольной задержки первична и не формализуется. Различаются только динамическая задержка, которая является функцией вхождения вычислительного акта в историю вычисления, и статическая задержка, которая является функцией вхождения предписания о вычислительном акте в текст программы.

Задержка вычислительного акта может повлечь за собою серию вынужденных задержек. Правила вынужденной задержки уже полностью детерминированы и гарантируют корректность смешанного вычисления. В этом изложении, однако, мы дадим лишь правдоподобное приближение к точным правилам смешанного вычисления, которые, в применении к алголоподобным программам могут быть найдены в [2].

П1) Задержанный подтерм t вынужденно задерживает терм, для которого t является прямой конституентой.

П2) Задержанный терм T вынужденно задерживает оператор, для которого T является прямой конституентой.

П3(динамика). Задержанный оператор $x:=T$ (аналогично для оператора ввода) задерживает любой терм, который использует результат данного присваивания.

П3 (статика). Задержанный оператор $x:=T$ задерживает любой терм, который, в принципе, мог бы использовать результат данного присваивания.

П4) Задержанный составной оператор (условный, цикл) вынужденно задерживает все операторы, являющиеся его прямыми конституентами.

Замечание I. Задержка составного оператора не обязательно сопровождается задержкой его предикатного терма.

241-133

Замечание 2. Поскольку любой терм является конституентой некоторого оператора, можно считать, что понятие вынужденной задержки есть бинарное отношение между операторами (оператор S_1 вынужденно задерживает оператор S_2), а само свойство задержанности (для статики лучше сказать задерживаемости) является транзитивным.

Опишем теперь для любой программы P смешанное вычисление M(P) в языке МИЛАН. Мы не будем формализовать понятие состояния памяти, а опишем лишь способ формирования остаточной программы (в начальный момент остаточная программа пуста). Очевидно, что правила смешанного вычисления будут рекурсивны.

M(< оп >; < серия >) = M(< оп >); M(< серия >).

Если < оп > не задержан, то

M(< оп >) = V (< оп >).

Пусть < терм > ::= < вып > | < вып > < усл > < вып > .

Если < терм > не задержан, то

M(< терм >) = V(< терм >).

Если < терм > задержан, то

M(< терм >) = < терм 1 >, где < терм 1 > получается из < терм > вычислением всех незадержанных подтермов и заменой их вхождений на изображения их значений.

Ниже все операторы считаются задержанными, а результатом для M является литеральное значение оператора, приписываемое к остаточной программе.

M(< пер > := < выр >) = < пер > := M(< выр >).

241-134

М(если р то Сер1 иначе Сер2 все) =

$$= \begin{cases} \text{если } М(р) \text{ то } М(Сер1) \text{ иначе } М(Сер2) \text{ все}, & \text{если р задержан;} \\ М(Сер1), & \text{если } р = \text{ист;} \\ М(Сер2), & \text{если } р = \text{ложь.} \end{cases}$$

М(пока р цк Сер кц) =

$$= \begin{cases} \text{пока } М(р) \text{ цк } М(Сер) \text{ кц}, & \text{если р задержан;} \\ М(Сер);\ М(\text{пока } р \text{ цк } Сер \text{ кц}), & \text{если } р = \text{ист;} \\ \Phi \quad (\text{пусто}), & \text{если } р = \text{ложь.} \end{cases}$$

Замечание. Остаточная программа может быть как короче (редукция термов и альтернатив), так и длиннее (раскрутка циклов) исходной программы. Однако она будет всегда быстрее исходной программы благодаря разгрузке операторов и сокращению накладных расходов в составных операторах.

I.3. Генерирующее расширение в языке МИЛАН. В общем случае смешанное вычисление требует прокрутки, т.е. привлечения к процессу "чистого" выполнения программы некоторого механизма интерпретации. В ряде случаев, когда задерживаемость устанавливается статически можно определить некоторое "генерирующее расширение" G программ P, такое, что для любого X имеет место функциональное равенство

$$M(P,X) = V\ (G(P),X)\ .$$

Определим генерирующее расширение в языке МИЛАН. Пусть в P помечены произвольно задерживаемые термы и операторы. Основываясь на статическом определении отношения вынужденной задерживаемости, возьмем его транзитивное замыкание, что даст нам полное множество задерживаемых термов и операторов. Введем в язык МИЛАН переменную T, значением которой являются тексты программ (в ней будет на-

241-135

называться остаточная программа) и операцию транс, аргументом которой являются параметризованные записи операторов МИЛАНа или их фрагменты. Параметры – это особые переменные n1, n2,... , значениями которых являются записи констант. Оператор вида n1:=t , вычисляя терм t , осуществляет приведение значения t к изображению этого значения. Оператор вида T:=транс (S(n1, n2);) приписывает к текущему значению T запись (литеральное значение) оператора S с предварительной заменой n1 и n2 на их значения. Например, если текущие значения термов a, a+b и t + 3 × d равны, соответственно, 0, 4 и 57, то в результате выполнения операторов

```
n1 := a ;
n2 := a + b ;
n3 := t + 3 x d ;
T:=транс (если n1 < x то z := n2 + y иначе z:= n3 все;)
```

к остаточной программе будет приписан следующий текст:

если 0 < x то z:=4 + y иначе z :=57 все ; .

Определим генерирующее расширение G в МИЛАНе, задав для каждого оператора S программы P его образ в G (P). Для определенности будем считать, что в каждом задерживаемом терме есть два незадерживаемых подтерма s и t .

G (<оп>; <серия>) = G (<оп>); G(<серия>) .

Если <оп> не задерживаем, то

G (<оп>) = <оп>.

Ниже все операторы считаются задерживаемыми.

```
G (y:=F(s,t)) =
  = n1:=s; n2:=t; T:= транс (y:=F(n1,n2);) .
G (если p( s , t ) то Сер1 иначе Сер2 все) =
```

$$= \begin{cases} (\underline{\text{если}}\ p\ (s\ ,\ t\ )\ \underline{\text{то}}\ G(\text{Сер1})\ \underline{\text{иначе}}\ G\ (\text{Сер2})\ \underline{\text{все}}, \text{ если } p \text{ не задерживаем;} \\ \left.\begin{array}{l} n1\ :=\ s;\ \ n2:=\ t\ ;\ T:=\text{транс}\ (\underline{\text{если}}\ p(n1,n2)\ \underline{\text{то}}); \\ G\ (\text{Сер1});\ T:=\text{транс}\ (\underline{\text{иначе}});\ G\ (\text{Сер2}); \\ T:=\ \text{транс}\ (\underline{\text{все}};) \end{array}\right\} \text{, если } p \text{ задерживаем.} \end{cases}$$

$$G\ (\underline{\text{пока}}\ p(s\ ,t\ )\ \underline{\text{цк}}\ \text{Сер}\ \underline{\text{кц}}) =$$

$$= \begin{cases} \underline{\text{пока}}\ p(s\ ,t\ )\ \underline{\text{цк}}\ G\ (\text{Сер})\ \underline{\text{кц}} & \text{, если } p \text{ не задерживаем;} \\ \left.\begin{array}{l} n1\ :=\ s;\ \ \ n2:=\ t;\ T:=\text{транс}\ (\underline{\text{пока}}\ p(n1\ ,n2\ )\ \underline{\text{цк}}) \\ G\ (\text{Сер});\ T:=\text{транс}\ (\underline{\text{кц}};) \end{array}\right\} \text{, если } p \text{ задерживаем.} \end{cases}$$

В качестве начального оператора $G\ (P)$ берется присваивание пустого слова переменной T .

Если выполнить на начальном состоянии памяти X построенное таким образом генерирующее расширение $G\ (P)$, мы получим промежуточное состояние памяти X , дополнительно к нему какие-то значения переменных $n1$, $n2$, а в качестве значения переменной T – остаточную программу P' , удовлетворяющую принципу частичного вычисления.

Ниже дается генерирующее расширение программы 1 :

Программа 5

```
T:= ∅; T:=транс(y:=1;); пока n >0 цк если нечет (n ) то
   T:=транс(y:=y * x;); n := n -I все ;
  n := n+2 ; T:=транс(x:=x * x;) .
```

В результате ее выполнения для $n = 5$ получится следующая остаточная программа:

Программа 6

```
y:=1; y:=y * x; x:=x * x; x:=x * x; y:=y * x; x:=x * x;.
```

241-137

Интересно сравнить между собой программы 2,4 и 6 :

```
(2):      y:=x;     x:=x * x; x:=x * x; y:=y * x
(4):      y:=1 * x; x:=x * x; x:=x * x; y:=y * x; x:=x * x
(6): y:=1; y:=y * x; x:=x * x; x:=x * x; y:=y * x; x:=x * x;
```

(4) - (6) демонстрируют разницу между динамическими и статистическими задержками (последние, естественно, грубее). (2) - (4) показывает, что остаточная программа, как правило, требует некоторой дальнейшей оптимизации, как локальной (устранение умножения на 1), так и глобальной (устранение неиспользуемых вычислений).

I.4. Важный частный случай. Пусть начальное состояние памяти W программы P в языке L каким-то образом разбито на две составляющие X и Y :

$$W = (X,Y) ,$$

каждую из которых можно рассматривать отдельно. В качестве единственной причины произвольной задержки возьмем замораживание части Y , т.е. задерживается любой терм, содержащий аргументом величину из Y . В этом случае все последующие задержки будут вынужденными и весь процесс смешанного вычисления будет полностью детерминированным. В точном формализме можно показать, что в этом случае остаточная программа, естественно, оказывается функцией X :

$$M_G (P, (X,Y)) = F(X) , \qquad (1)$$

а промежуточное состояние памяти - это просто

$$M_C (P, (X,Y)) = Y . \qquad (2)$$

Другими словами, остаточная программа вбирает в себя полностью всю зависимость вычислений от части X . Принцип частичных вычислений приобретает в этом случае форму следующего функционального

241-138

равенства

$$V\ (P,\ (X,Y)) = V\ (M_G\ (P,X),\ Y)\ , \qquad (3)$$

показывающего, что левое общее вычисление может быть реализовано как расчлененное вычисление: сначала смешанное вычисление по исходной программе, а затем – вычисление по остаточной программе. Само утверждение о равенстве можно назвать теоремой о факторизации.

В условиях теоремы о факторизации можно формально описать общую процедуру построения генерирующего расширения G (P). Напомним, что, по определению, для генерирующего расширения справедливо функциональное равенство

$$M\ (P,W) = V\ (G(P),W)\ . \qquad (4)$$

Пусть в языке L запрограммирована процедура смешанного вычисления, т.е. существует программа M^*, удовлетворяющая функциональному равенству

$$M\ (P,W) = V\ (M^*,(P,W))\ . \qquad (5)$$

Подвергнем программу M^* на начальном состоянии памяти (P, W) с замороженным W, в свою очередь, смешанному вычислению. В соответствии с (I) и (2) мы получим, что

$$M_G\ (M^*,(P,W)) = F(P)\ , \qquad (6)$$

$$M_G\ (M^*,(P,W)) = W\ . \qquad (7)$$

По определению остаточной программы (3) и программы M^* (5), мы получим:

$$V\ (F(P),W) = V\ (M^*,(P,W)) = M(P,W)\ . \qquad (8)$$

Сравнивая (4) и (8), мы получаем, что для любых P и W справедливо следующее функциональное равенство

$$V\ (G(P),W) = V\ (F(P),W)\ . \qquad (9)$$

241-139

Поскольку повторное применение смешанного вычисления само может быть выполнено применением программы M^*, способ получения генерирующего расширения можно схематично записать в виде следующего равенства

$$G(P) = M^*(M^*(P)). \qquad (10)$$

Это утверждение, предложенное В.Ф.Турчиным (см. ОБСУЖДЕНИЕ), может быть названо теоремой о двойной прогонке.

2. ТРАНСЛЯЦИЯ

2.1. Общий тезис. Программа, как правило, выполняется многократно на множестве исходных данных. Эти данные обычно имеют некоторую структуру, например состоят из двух частей X и Y. В разнообразие исходных данных разные части в общем случае также вносят неодинаковый вклад, когда на фиксированное значение части X приходится много значений Y. В таких случаях естественно программу как функцию двух "переменных" адаптировать к заданному X, рассматривая ее по отношению к Y как частную функцию одного переменного.

Для системного программирования характерной является именно такая ситуация, когда глобальный многопараметрический вычислительный процесс (например, прохождение программы через мощную вычислительную систему общего назначения) желательно приспособить к суженному, по устойчивому контексту, в котором предполагается многократно и эффективно использовать данную программу. На всех этажах системного программирования происходит постоянная борьба универсальности и специализации, а для поисков оптимальных выходов из постоянно возникающих противоречий предложено большое количество весьма разных методов и приемов. Есть основание полагать, что многие из них являются ни чем иным как специальными выражениями смешанных вычислений.

241-140

Смешанные вычисления, по-видимому, могут единым образом описывать функционирование разнообразных генераторов программ, применяемых в системном программировании: компоненты языковых процессоров (сканнеры, анализаторы, генераторы кода), конфигураторы, комплексаторы и макрогенераторы в системах модульного программирования. Все процессы, так или иначе связанные с адаптацией универсальных компонент к заданным параметрам (грамматика или семантика конкретного языка, параметры оборудования, запросы к обстановке, размерность задачи, объем или размещение данных), могут в этом сложном хозяйстве выполняться однообразно с помощью процедуры смешанного вычисления.

Автор постарается обосновать этот тезис, рассмотрев некоторые существенные компоненты программирующих процессоров.

2.2. Интерпретационная семантика как источник объектного кода. Рассмотрим семантику некоторого алгоритмического языка. В ее операционном выражении – это некоторый универсальный алгоритм S (интерпретатор), который для программы P (например, заданной ее деревом разбора) и входных данных X задает ее вычисление: $Y = S(P,X)$. Это вычисление, в свою очередь, реализуется универсальным вычислением V в языке реализации: $Y = V(S,(P,X))$. Можно показать, что, замораживая X в смешанном вычислении $M(S,(P,X))$, мы получим в качестве остаточной программы объектный код программы P , применение которого к X даст то же вычисление

$$S'_P(X) = S(P,X) = Y .$$

Взяв в качестве языка реализации Алгол 68, рассмотрим абстрактный синтаксис языка МИЛАН, представленный совокупностью описания видов:

241-141

синтаксис языка МИЛАН

вид программа = ст (серия телпр);
вид серия = ст (имя опер очер, имя серия хвост);
вид опер = об (присв, чит, выв, усл, цикл);
вид присв = ст (пер получ, имя выр источн);
вид выр = об (бином, пер, конст);
вид чит = ст (сепр получ);
вид сепр = ст (пер очер, имя сепр хвост);
вид выв = ст (сепр источн);
вид усл = ст (отн усл, имя серия тооп, иноп),
вид цикл = ст (отн загл, имя серия телц);
вид отн = ст (имя выр лев, прав, строк опер, имя лог знач);
вид бином = ст (имя выр лев, прав, строк опер, имя цел знач);
вид пер = ст (имя строк предст, имя цел знач);
вид конст = пер .

Интерпретационная семантика языка МИЛАН также заимствована из [1] с некоторыми модификациями, предложенными в [3]. В частности, объектная часть программы – переменные и константы – не выделяется в вектор состояний, а образует терминальные вершины дерева программы. Во время генерации значения объектов считаются недоступными, но их имена – известными. Примитивами объектного кода считаются символические трехадресные команды, представленные в одежде Алгола 68.

241-142

Синтаксис языка "МИЛАН"

проц ПРОГРАММА = (программа ПР) пуст: СЕРИЯ (телпр из ПР);

проц СЕРИЯ = (имя серия СЕР) пуст:

пока ОПЕР(очер из СЕР); хвост из СЕР ≠ нил цк СЕР:=хвост из СЕР кц ;

проц ОПЕР = (имя опер ОР) пуст:

выб ОР в (присв ОР) : ПРИСВ(ОР),

(чит ОР) : ЧИТ(ОР),

(выв ОР) : ВЫВ(ОР),

(усл ОР) : УСЛ(ОР),

(цикл ОР): ЦИКЛ(ОР) быв;

проц ПРИСВ = (имя присв ПР) пуст :

(ВЫР(источн из ПР); знач из получ из ПР:=знач из источн из ПР);

проц ЧИТ = (имя чит Ч) пуст :

пока ЗАГР(знач из очер из получ из Ч);хвост из получ из Ч≠нил

цк получ из Ч:=хвост из получ из Ч кц;

ц проц ЗАГР/ВЫГР – базисные процедуры ввода/вывода ц ,

проц ВЫВ = (имя выв В) пуст :

пока ВЫГР(знач из очер из источн из В);хвост из источн из В≠нил

цк источн из В:=хвост из источн из В кц;

проц УСЛ = (имя усл У) пуст :

начало ОТН(усл из У); если ¬ знач из усл из У то на М1 все;

СЕРИЯ(тоор из У); на М2; М1: если иноп из У=нил то на М2 все;

СЕРИЯ(иноп из У); М2: конец;

проц ЦИКЛ = (имя цикл Ц) пуст:

начало М : ОТН (загл из Ц);

если ¬ знач из загл из Ц то на М2 все;

СЕРИЯ (телц из Ц); на М1; М2: конец;

241-143


```
проц ОТН = (имя отн ОН) пуст :
    начало ВЫР (лев из ОН); ВЫР (прав из ОН);
    знач из ОН:=опер из ОН = "≺ " ∧
                    знач из лев из ОН ≺ знач из прав из ОН ∨
                    опер из ОН = "≼" ∧
                    знач из лев из ОН ≼ знач из прав из ОН ∨
                    опер из ОН = "=" ∧
                    знач из лев из ОН = знач из прав из ОН ∨
                    опер из ОН = "≽" ∧
                    знач из лев из ОН ≽ знач из прав из ОН ∨
                    опер из ОН = "≻" ∧
                    знач из лев из ОН ≻ знач из прав из ОН ∨
                    опер из ОН = "≠" ∧
                    знач из лев из ОН ≠ знач из прав из ОН конец;
проц ВЫР = (имя вид ВК) пуст :
    выб ВК в (бином ВК) : БИНОП(ВК)
                иначе скип быр;
проц БИНОП = (имя бином БО) пуст :
    начало ВЫР(лев из БО); ВЫР(прав из БО);
    знач из БО:= если опер из БО = "+" то
                        знач из лев из БО - знач из прав из БО
                 иначе опер из БО = "-" то
                        знач из лев из БО - знач из прав из БО
                 иначе опер из БО = "×" то
                        знач из лев из БО × знач из прав из БО
                 иначе знач из лев из БО ÷ знач из прав из БО
    все конец.
```

241-144

Рассмотрим программу в языке МИЛАН, также взятую из [1].

Программа 7

```
нач
   чит n ;
    i := 0;
   пока  i < n цк
      i := i + 1;
      чит x,y;
      если  x<0 то
         z := x +(5 * y);
         вып z  все кц;
   вып z
кон.
```

Дерево абстрактного синтаксиса этой программы показано ниже (мнемонические названия служат в качестве имен внутренних ссылок, ххх - замороженное значение).

Программа 8

```
программа = ((ввод n, прод1))
ввод n = ((("n", знач n), нил))
знач n = ххх
прод1 = ( i присв0, прод2)
 i присв0 = (("i", знач i), нуль)
знач i = ххх
нуль =("0", знач0)
знач0 = ххх
прод2 = (цикл по i , прод3)
цикл по i = ((лев i , прав n, "<", т1), тело)
лев i = ("i", знач i)
прав n = ("n", знач n)
т1 = ххх
тело = (счет i , прод21)
счет i = (("i", знач i ), i плюс1)
 i плюс1 = (выр i , выр1,  "+", т2)
выр i = ("i", знач i)
выр1 = ("1", знач1)
```

241-145


```
знач1 = ххх
ч2 = ххх
прод21 = (ввод ху, прод22)
ввод ху = ((("х", знач х), след у))
знач х = ххх
след у = (("у", знач у), нил)
знач у = ххх
прод22 = (услов, нил)
услов = ((лев х, прав0, "<", ч3), то серия, нил)
лев х = ("х", знач х)
прав0 = ("0", знач0)
ч3 = ххх
то серия = (счет ч , прод221)
счет ч = (("ч", знач ч), форм1)
знач ч = ххх
форм1 = (выр х, форм2, "+", ч5)
выр х = ("х", знач х)
форм2 = (выр5, выр у, "х", ч4)
выр5 = ("5", знач5)
знач5 = ххх
выр у = ("у", знач у)
ч4 = ххх
ч5 = ххх
прод221 = (вывод ч1 , нил)
вывод ч1 = ((("ч", знач ч), нил))
прод3 = (вывод ч2 , нил)
вывод ч2 = ((("ч", знач ч), нил)) .
```

Семантику языка МИЛАН можно подвергнуть смешанному вычислению над программой 6, заморозив в ней значения констант и переменных. Очевидно, что правила смешанного вычисления в Алголе 68 несколько сложнее, нежели в МИЛАНе: при выполнении процедур происходит копирование локальных объектов, в частности, меток. В результате получается следующий объектный код.

241-146


```
    ЗАГР (знач n);
    знач i := знач0;
M1: τ1 := знач i < знач n;
    если ¬ τ1 то на M2 все;
    τ2 := знач i + знач1;
    знач i := τ2;
    ЗАГР (знач x);
    ЗАГР (знач y);
    τ3 := знач x < знач0;
    если ¬ τ3 то на MM все;
    τ4 := знач5 × знач y;
    τ5 := знач x + τ4 ;
    знач z := τ5;
    ВЫГР (знач z);
MM : на MM2;
MM2 : на M1;
 M2 : ВЫГР (знач z).
```

Итак, мы получаем

Заключение 1

ГЕНЕРАЦИЯ ОБЪЕКТНОГО КОДА ПРИ ТРАНСЛЯЦИИ ПРОГРАММЫ P ВХОДНОГО ЯЗЫКА L - ЭТО СМЕШАННОЕ ВЫЧИСЛЕНИЕ ИНТЕРПРЕТАЦИОННОЙ СЕМАНТИКИ ЯЗЫКА L НАД ПРОГРАММОЙ P С ЕЕ ЗАМОРОЖЕННОЙ ОБЪЕКТНОЙ ЧАСТЬЮ .

Замораживая в процессе смешанного вычисления семантики $M(S,(P,X))$ не только X , но также и разные конструкции программы P , мы получим разные варианты объектного кода, отличающиеся глубиной интерпретации программы - от чистой интерпретации, когда P задержано целиком, до чистой трансляции, когда задержано только X . Наоборот, размораживая отдельные компоненты

241-147

из X ... мы будем получать разные версии трансляции, автоматически подстраиваемые к заданным заранее параметрам объектного вычисления. Таким образом, можно сделать

Заключение 2

ПРОЦЕССОР СМЕШАННЫХ ВЫЧИСЛЕНИЙ В ЯЗЫКЕ РЕАЛИЗАЦИИ ПОЗВОЛЯЕТ ДЛЯ ЕДИНЫМ ОБРАЗОМ ЗАДАННОЙ СЕМАНТИКИ ВХОДНОГО ЯЗЫКА ПОЛУЧАТЬ ШИРОКОЕ РАЗНООБРАЗИЕ СХЕМ ТРАНСЛЯЦИИ, УПРАВЛЯЕМОЕ СТЕПЕНЬЮ ДОСТУПНОСТИ РАЗЛИЧНЫХ КОМПОНЕНТ ВХОДНОЙ ПРОГРАММЫ .

2.3. Систематическое построение генератора объектного кода. Для семантики S языка L можно построить генерирующее расширение $G(S)$. Сделаем это для семантики языка МИЛАН. Ниже Т и транс сохраняют смысл п. 1.3; все литеральные переменные начинаются с буквы n . Правила генерирующего расширения дополняются операциями копирования локальных объектов (меток) и чем-то вроде условной компиляции. В результате получается трансляционная семантика языка МИЛАН

```
проц ПРОГРАММА = (программа ПР) пуст : СЕРИЯ (телпр из ПР);
проц СЕРИЯ = (имя серия СЕР) пуст :
   пока ОПЕР(очер из СЕР); хвост из СЕР≠нил цк СЕР:=хвост из СЕР
                                                        кц;

проц ОПЕР = (имя опер ОР) пуст :
   выб ОР в (присв ОР) : ПРИСВ (ОР),
            (чит ОР) : ЧИТ(ОР),
            (выв ОР) : ВЫВ(ОР),
            (усл ОР) : УСЛ(ОР),
            (цикл ОР): ЦИКЛ(ОР) быв;
```

241-148


```
проц ПИШВ = (имя прим ПР) пуст :
    (ВЫР(источн из ПР); n1 := знач из получ из ПР;
    n2 :=знач из получ из ПР; Т:=транс( n1 := n2;));
проц ЧИТ = (имя чит Ч) пуст :
    пока n1 :=знач из очер из получ из Ч; Т:=транс(ЗАГР( n1 ););
        хвост из получ из Ч ≠ нил
     цк получ из Ч:=хвост из получ из Ч кц;
п  ЗАГР/ВЫГР - базисные процедуры ввода/вывода  п
проц ВЫВ = (имя выв В) пуст :
    пока n1 :=знач из очер из источн из В; Т:=транс(ВЫГР( n1 ););
        хвост из источн из В ≠ нил
     цк источн из В:=хвост из источн из В кц;
проц УСЛ = (имя усл У) пуст : начало n М1 :=копия(М1 );
                                     n М2:=копия(М2);
    ОТН(усл из У); n1 :=знач из усл из У; Т:=транс(если ¬ n1 то
                                                  на n М1 все;);
    СЕРИЯ(тоол из У); Т:=транс(на n М2); М1: Т:=транс( n М1 :);
    если инол из У=нил то на М2 все; СЕРИЯ(инол из У);
    М2: Т:=транс( n М2:) конец;
п  копия - служебная процедура, создающая новый экземпляр
   локального объекта  п
проц ЦИКЛ = (имя цикл Ц) пуст:
    начало n М1 :=копия(М1 ); n М2:=копия(М2); М1: Т:=транс( n М1 );
    ОТН(загл из Ц); n1 :=знач из загл из Ц;
    Т:=транс(если ¬ n1 то на n М2 все;);
    СЕРИЯ(телц из Ц); Т:=транс(на n М1;);
    М2: Т:=транс( n М2:) конец;
```

241-149


```
проц ОТН = (имя отн ОН) пуст :
    начало ВЫР(лев из ОН); ВЫР(прав из ОН);
      n1:=знач из ОН; nлев:=знач из лев из ОН;
      nправ:=знач из прав из ОН;
      Т:=транс(n1:=ускомп (
              опер из ОН = "<" ∧ (nлев < nправ)∨
              опер из ОН = "≤" ∧ (nлев ≤ nправ)∨
              опер из ОН = "=" ∧ (nлев = nправ)∨
              опер из ОН = "≥" ∧ (nлев ≥ nправ)∨
              опер из ОН = ">" ∧ (nлев > nправ)∨
              опер из ОН = "≠" ∧ (nлев ≠ nправ));) конец;
```

п уском – служебная процедура условной компиляции п

```
проц ВЫР = (имя выр ВК) пуст :
    выб ВК в (бином ВК): БИНОП(ВК)
              иначе скип быв;

проц БИНОП = (имя бином БО) пуст :
    начало ВЫР (лев из БО); ВЫР(прав из БО);
    n1:=знач из БО; nлев:=знач из лев из БО;
    nправ:=знач из прав из БО;
    Т:=транс (n1:=ускомп (
          если опер из БО = "+" то (nлев + nправ)
          инес опер из БО = "-" то (nлев - nправ)
          инес опер из БО = "*" то (nлев * nправ)
                               инес  (nлев ÷ nправ)
          все);) конец;
```

Если, в обозначениях предыдущего пункта, применить G(S) к входной программе P , то результатом будет объектный код S'_P программы P . Таким образом получаем

Заключение 3

ПРАВИЛА ГЕНЕРИРУЮЩЕГО РАСШИРЕНИЯ ДАЮТ СИСТЕМАТИЧЕСКУЮ ПРОЦЕДУРУ ПРЕОБРАЗОВАНИЯ ИНТЕРПРЕТАЦИОННОЙ СЕМАНТИКИ ВХОДНОГО ЯЗЫКА В ГЕНЕРАТОР ОБЪЕКТНОГО КОДА.

Сравнивая правила построения генерирующего расширения с разнообразными правилами макрогенерации можно сделать

Заключение 4

ОПЕРАТОРЫ ПЕРИОДА КОМПИЛЯЦИИ - ЭТО СИНТАКСИЧЕСКИ УЗАКОНЕННЫЕ СМЕШАННЫЕ ВЫЧИСЛЕНИЯ, ДОПУСКАЕМЫЕ В ТРАНСЛЯТОРЕ. МАКРОГЕНЕРАЦИЯ И ПОСТРОЕНИЕ ГЕНЕРИРУЮЩЕГО РАСШИРЕНИЯ РЕАЛИЗУЮТСЯ ПО СУЩЕСТВУ ОБЩИМИ ПРИЕМАМИ.

2.4. Смешанные вычисления и оптимизация. Замечательной особенностью смешанных вычислений является то, что они выбирают из исходной программы и складывают в остаточную программу только те исполнительные команды, которые канизываются на единственную динамическую нить вычислений, предписываемых начальным состоянием памяти. При этом также выполняются такие управляющие действия как проверки условий, вызовы процедур и передачи параметров. Наконец, многие термы заменяются своими значениями. Недостаточно осознанная имитация этих вычислительных актов в трансляторах привела к большому количеству особого рода оптимизаций (см., например, [4] и [5]).

241-151

Выполнение константных действий (в том числе и в предикатных термах), открытая подстановка процедур, раскрытие циклов – все эти виды оптимизации являются частными случаями смешанных вычислений.

Чистка циклов с телом B основана на разбиении данных цикла X на две части: X_S – данные, которые не меняются при выполнении B и X_D – переменная часть. Применяя к B смешанное вычисление с замороженным X_D, получим остаточную программу – разгруженное тело цикла. Термы, опирающиеся на X_S, могут быть вычислены до входа в цикл. Итак, мы получаем

Заключение 5

ГЕНЕРАЦИЯ ОБЪЕКТНОГО КОДА С ПОМОЩЬЮ СМЕШАННЫХ ВЫЧИСЛЕНИЙ АВТОМАТИЧЕСКИ РЕАЛИЗУЕТ МНОГИЕ ОПТИМИЗИРУЮЩИЕ ПРЕОБРАЗОВАНИЯ .

Одной из трудных проблем реализации сложных алгоритмических языков является реализация смешанной стратегии программирования [6], или кейсинга (casing) [21]. Главная проблема – иметь в объектной программе неизбыточный код. Это требует при построении генератора кодов учитывать большое число частных случаев. В системе АЛЬФА [6] имелось 11 способов программирования нерекурсивных процедур, 58 способов реализации операции умножения. Работа [21] показывает, что обычная техника оптимизирующей трансляции с ПЛ/I требует генератора объектного кода, в 50 раз большего, чем для оптимизирующей трансляции с Фортрана.

Есть основания полагать, что смешанные вычисления благодаря своей естественной селективности могут автоматически ликвидировать большую часть избыточности объектного кода.

241-152

Рассмотрим фрагмент семантической рекурсивной процедуры, реализующей оператор присваивания многомерных величин в духе Алгола 68. Получатель может иметь фиксированные и гибкие границы. При присваивании получателю с фиксированными границами источник должен иметь такие же длины по измерениям. Исключение составляет источник-нуль, который может быть присвоен любой величине с очевидным приведением. При присваивании получателю с гибкими границами длины по измерениям берутся от источника. Если получатель имеет большую размерность, происходит укрупнение, когда избыточные измерения получателя получают единичные длины. Семантический контроль в процедурах опущен.

```
вид мульти = ст (база база п начальный адрес п, лог тип п ист -
     фиксированные длины, ложь - гибкие п, цел мерн п размерность п,
     список длин п по измерениям п, имя возмвид знач);
вид список = ст (цел очер, имя список след);
вид скаляр = ст (база база, имя возмвид знач);
вид конст = ст (база база, имя возмвид знач);
проц ПРИСВОЙМУЛЬТИЗНАЧ = (мульти ПОЛ п получатель п,
     об (мульти, скаляр, конст) ИСТ п источник п) пуст:
     начало
          проц передача = (имя список полинд, истинд п индексы
          получателя и источника п, мульти пол, об (мульти, скаляр,
          конст)ист, имя список длин, цел разм, дефект п разность
          размерностей получателя и источника п) пуст:
               если дефект > 0 то
                    передача (глоб список:=(1,полинд),истинд,пол,
                              ист, длин, разм-1, дефект-1)
```

241-153

инес разм>0 то для i до очер из длин цк

передача (глоб список:=(i ,полинд),глоб список:=(i ,
истинд),
пол, ист, длин:=след из длин,разм-1,0) кц

инес выб ист в (мульти ист): ПЕР(база из пол, полинд,
база из ист, истинд)
иначе ПЕР(база из пол,полинд,база из ист)
быв все;

п ПЕР - объектная команда передачи п;

если тип из ПОЛ то
передача (нил, нил, ПОЛ, ИСТ, длин из ПОЛ, мерн из ПОЛ,0)
иначе выб ИСТ в (мульти ИСТ):
передача (нил, нил, ПОЛ, ИСТ, длин из ИСТ, мерн из ПОЛ,
мерн из ПОЛ - мерн из ИСТ)
(конст ИСТ):
если знач из ИСТ=0 то
передача (нил, нил, ПОЛ, ИСТ, нил, мерн из ПОЛ, 0)
иначе передача (нил, нил, ПОЛ, ИСТ, нил, мерн из ПОЛ,
мерн из ПОЛ) все
иначе передача (нил, нил, ПОЛ, ИСТ, нил, мерн из ПОЛ,
мерн из ПОЛ) быв все
конец .

При построении объектного кода методов смешанных вычислений из этой прямолинейной семантической процедуры получаются весьма различные фрагменты объектного кода.

а) ПЕР(р,(1,нил),0); ПЕР(р,(2,нил),0); ПЕР(р,(3,нил),0)
п засылка нуля в вектор р(3) п ;

241-154


```
а) ПЕР(а,(1,(1,нил)),с,(1,(1,нил)));ПЕР(а,(1,(2,нил),с,
                                          (1,(2,нил)));
   ПЕР(а,(2,(1,нил)),с,(2,(1,нил)));ПЕР(а,(2,(2,нил),с,
                                          (2,(2,нил)))
                  п а(2,2):= с(2,2) п;
б) для i1 до n1 цк для i2 до n2 цк ПЕР(Т,(1,(1,(i1,(i2,нил)))),
                                       X,(i1,(i2,нил))) кц кц
```

п передача двумерного массива X в четырехмерный гибкий массив T п ;

г) передача (нил, нил, А,В, длин из В,5,5 – мерн из В)

п загрузка 5–мерного гибкого массива А многомерным формальным параметром В , который замещается фактическими параметрами разной размерности п .

Еще один пример приводится в разделе 3.5.

Итак мы имеем возможность сделать

Заключение 6

СМЕШАННЫЕ ВЫЧИСЛЕНИЯ АВТОМАТИЧЕСКИ ОБЕСПЕЧИВАЮТ ИЗБИРАТЕЛЬНОСТЬ В РЕАЛИЗАЦИИ СЛОЖНЫХ КОНСТРУКЦИЙ ВХОДНЫХ ЯЗЫКОВ.

3. ОБСУЖДЕНИЕ

3.1. Авторская линия. Автор пришел к идее смешанных вычислений, изучая роль универсальных оптимизирующих преобразований в трансляции. Еще в работе над системой АЛЬФА [6],[7] была понята разница между чисто комбинаторными оптимизационными преобразованиями и направленными преобразованиями (выполнение константных действий, чистка циклов, смешанная стратегия при реализации составных конструкций). Уже тогда было видно, что определенные виды направленной оптимизации требуют как некоторых частичных вычислений, так и определенного анализа информационных и логических связей.

Определенный прогресс был достигнут при разработке идеологической платформы проекта БЕТА [8] . В этой работе было на примере реализации процедур показано, что универсальные направленные преобразования в определенном сочетании с комбинаторной оптимизацией позволяют, по крайней мере в принципе, исключить смешанную стратегию программирования как независимую концепцию в разработке трансляторов. Другое наблюдение состояло в том, что все виды операционной активности во время трансляции и исполнения могут быть распределены по уровням, названным в работе участками повторяемости. Каждый участок повторяемости характеризуется разбиением доступных ему объектов на объекты с фиксированными (связанными) на данном участке значениями и на переменные на этом участке объекты. Действия, использующие связанные переменные могли тем самым переноситься на более редко исполняемые участки (разгрузка участков повторяемости). Была высказана гипотеза, что систематизация участков повторяемости, включающая период трансляции позволит создать "непрерывный спектр" реализаций языка -

от чистой интерпретации до чистой трансляции.

Только в прошлом году была найдена процедура "семантически управляемой генерации", т.е. систематического построения объектного кода из интерпретационной семантики языка. Сначала эта процедура описывалась как направленное применение нескольких универсальных оптимизаций (раскрытие процедур и циклов, выполнение константных действий, чистка программы от невыполняемых операторов). Этот промежуточный результат рассказывался автором на советско-американском семинаре по языкам программирования весьма высокого уровня (Москва, сентябрь 1976 г.) и в ряде докладов во время лекционной поездки по Великобритании (октябрь 1976 г.) [9] .

Концепция смешанных вычислений в том виде, как она была сформулирована автором в конце 1976 г., опубликована в [10] . Трактовка смешанных вычислений в настоящей работе уже использует некоторую обратную связь, полученную от реакции слушателей и анализа литературы.

Уже очевидно, что смешанные вычисления и как концепция и как рабочий инструмент касаются наиболее фундаментальных сторон трансляции и вычислений в алгоритмических языках. Поэтому было бы естественно обнаружить в литературе ряд сходных идей, высказанных или в общей форме, или хотя бы в качестве эмпирических правил. И действительно, выяснилось, что вопрос частичных и смешанных вычислений имеет приличную историю.

3.2. Линия Ломбарди. Л.Ломбарди был, по-видимому, первым, кто рассмотрел вопрос частичных вычислений с общих позиций. В начале 60-х годов, в период своей работы в Массачусетском технологическом институте Ломбарди выдвинул концепцию "пошагового

241-157

восприятия данных в условиях незамкнутого человеко-машинного взаимодействия" [11]. В его представлении, "такие концепции как "трансляция" и "период исполнения", которые до сих пор в разной степени как двигают так и тормозят вычислительную практику, исключаются из рассмотрения и заменяются унифицированной методологией взаимодействия процессоров с их обстановкой". Ломбарди излагает эту методологию как "новую единую философию проектирования как машин, так и языков, направленную на обеспечение в них следующих качеств:

... Способность к разумному применению алгоритмов над неполностью заданными аргументными значениями, выдавая в качестве результата функции отсутствующих аргументов, т.е. новые алгоритмы, способные к восприятию дополнительной информации об аргументных значениях; очевидно, что в качестве аргументных значений могут выступать сами алгоритмы.

... В своей основе такие процессоры воспринимают в качестве определений алгоритмов выражения или "формы", заполняя в них "пустые места" (или неизвестные) в зависимости от степени наличия доступных аргументов (или оставляют их неизменными); заменяют обозначения форм соответствующими формами (опять-таки в зависимости от степени доступности); упрощают результат, выполняя базисные арифметические или логические действия."

Более подробное описание проекта содержится в [12].

Идеи Ломбарди получили существенное развитие в СССР. Г.Х.Бабич построил экспериментальную систему ДекАС (Декларативный язык для Ассоциативной обработки Списков) для ЭВМ Минск 22 [13], которая со временем переросла в систему Инкол (INcremental

Computation Language). Общие свойства этих систем объяснени авторами следующим образом [I4] :

"I) выполнение программы представляет собой формирование текста решения. При этом основной операцией является подстановка вместо неизвестной величины ее значения, если оно задано. Если же это значение не задано, то неизвестная величина остается в тексте решения без изменения;

2) после подстановки при выполнении так называемых пороговых условий производятся вычисления. Если эти условия не выполнены, то текст, описывающий вычислительные операции, остается в решении без изменения;

3) имеется возможность выполнить вычисление функции, когда заданы значения не всех аргументов, при этом выполняются только те действия, которые разрешены пороговыми условиями;

4) нет различия между языками исходных данных и программ;

5) язык рассчитан на интерпретацию, что в совокупности с операцией подстановки обеспечивает возможность вычислений с неполной информацией."

3.3. Линия Рефала. В 1966 г. В.Ф.Турчин описал универсальный язык символических преобразований Рефал, с самого начала задуманный как средство реализации различных алгоритмических языков, опирающееся на их интерпретационную семантику [I5] . Аналогично нормальным алгоритмам Маркова, основной операцией языка является подстановка (конкретизация в Рефале), а поиск очередной формулы подстановки требует сопоставления с образцом, находящимся в поле зрения Рефал-машины. В то же время, аналогично Лиспу, данные могут быть структурированы с помощью скобок. Ядро языка

241-159

не содержит операторов перехода по метке и операторов присваивания по имени переменной. В то же время допускаются имена функций и макроконструкций. Язык Рефал реализован для нескольких ведущих моделей ЭВМ.

В работе [16] была описана серия эквивалентных преобразований алгоритмов и функций, изображаемых программами на Рефале. Одно из этих преобразований, названное прогонкой, применяется к программам, содержащим свободные переменные, и состоит в выполнении в программе всех конкретизаций, не требующих значений свободных переменных.

В этой работе было также указано, что прогонка имеет большое значение в "теории компиляции". Рефал, строго говоря, является метаязыком и не имеет предметной области. Он работает над структурируемыми абстрактными объектами, представляемыми базисными символами языка. Правила Рефала, осуществляя селекцию и замещение фрагментов программы, лишь сводят одни объекты к другим. Для описания на Рефале семантики какого-то языка L нужно к рефал-программе S , обеспечивающей упорядоченный доступ к терминальным объектам языка, добавить конкретизирующие их примитивные семантические функции, выраженные в объектном коде. Объявив некоторые терминальные объекты дерева разбора программы P языка L (т.е. ее входные данные) свободными переменными, можно осуществить прогонку программы S , что даст трансляцию программы P в объектный код. Если запрограммировать алгоритм прогонки на Рефале, а затем полученную программу R подвергнуть прогонке, имея в виду ее применение к S , то результатом должен стать транслятор языка L в объектный код. В этом и состояла начальная идея вышеупомянутой теоремы о двойной прогонке.

241-160

3.4. Линия Лиспа. Лисп (если не говорить о машинных языках) был первым языком программирования с единой формой представления данных и програм. Это свойство языка было сразу отмечено Ломбарди, которым вместе с Рафаэлем[17] описал в 1966 году модификацию Лиспа, позволяющую выполнять частичные вычисления.

В последующие годы концепция частичных вычислений в Лиспе снова всплыла на поверхность в связи с построением интегрированных (т.е. содержащих как данные, так и программы) без данных в задачах искусственного интеллекта. В 1974 г. в лаборатории проф. Э.Сандевала сложилась концепция "частичного вычислителя",[18] с его использованием и в системах вычислений с неполной информацией, так и для упрощения програм при их конструировании в растущей базе данных[19]

Автор располагает еще двумя, правда, косвенными ссылками на не - давнюю работу П.Хендерсона и Дж.Морриса, в которой описывается "ленивый вычислитель" [20] , т.е. схема вычислений, обладающая, по свидетельству Дж.Маккарти, способностью "откладывать на потом" некоторые вычислительные операции, а также на очень интересную, судя по названию, работу И.Футамуры [25].

3.5. Линия ЭКС. В Уотсоновском исследовательском центре ИБМ "группа ЭКС" (группа Экспериментальной Компилирующей Системы) уже в течение ряда лет ведет исследовательский проект, направленный на построение эффективных трансляторов для языков нового поколения. По сравнению с начальной публикацией [21] общее направление работ претерпело ряд модификаций и впоследствии автору было интересно обнаружить определенную методологическую общность проекта ЭКС [22] и проекта БЕТА [23]: (ориентация на сложные входные языки, схематоподобный внутренний язык, интенсивное использование универсальных оптимизирующих преобразований, опирающихся на гло-

241 - 161

бальный анализ информационных связей). Сущность подхода группы ЭКС к проблеме трансляции лучше всего раскрывается следующей выдержкой из работы [22] .

"Генераторы кода – это главный механизм, с помощью которого транслятор сопоставляет семантику входного языка и семантику объектной машины. Для разработчика транслятора нужно указать по крайней мере одну "самую общую" конструкцию, реализующую операцию входного языка. Это в точности та спецификация, которая могла бы быть использована интерпретатором операции. Однако, такое общее определение игнорирует многие возможности для оптимизации. Эти возможности появляются из-за того, что контекст, в котором фактически применяется операция, может опираться на информацию, позволяющую интерпретировать некоторые части объектного кода в момент трансляции.

В нашем трансляторе семантика операции входного языка выражается обычной процедурой, обращение к которой в период исполнения всегда обеспечивает нужный результат. Было бы неразумным накладные расходы интерпретации обращения к данному генератору кода оставлять в коде, генерируемым оптимизирующим транслятором. С использованием раскрытия процедур, состоящего во вставке текста вызванной процедуры в место вызова, можно избежать избыточных накладных расходов интерпретации в период исполнения."

В работе [24] дается пример оптимизации объектного кода, соответствующего фрагменту программы в языке PL/I , выполняющему конкатенацию двух строк

```
P : PROC OPTIONS (MAIN) ;
    DCL (B,C) CHAR(10), A CHAR(50) ;
    A = B||C ;
    END ;
```

241-162

Буквальное выполнение семантических программ конкатенации, переноса строки и заполнения строки заданной литерой требует кода, состоящего из 55 инструкций внутреннего языка и выполнения 73 инструкций. Оптимизация (содержащая раскрытие процедур, выполнение константных действий, чистку программы и экономию величин) приведет к коду следующего вида

```
A = B (9 + 1 BYTES)
(:A: + 10) = C (9 + 1 BYTES)
(:A: + 20) = blank (0 + 1 BYTES)
(:A: + 21) = (:A: + 20) (28 + 1 BYTES) .
```

Применение смешанных вычислений к этим же процедурам даст следующий фрагмент остаточной программы

```
T1 = B (9 + 1 BYTES)
T2 = C (9 + 1 BYTES)
A = T1(9 + 1 BYTES)
(:A: + 10) = T2 (9 + 1 BYTES)
(:A: + 20) = blank (0 + 1 BYTES)
(:A: + 21) = (:A: + 20) (28 + 1 BYTES) .
```

Как и следовало ожидать, смешанное вычисление успешно реализует все указанные оптимизации, кроме , естественно, экономии величин.

3.6. Заключение. Мы не будем вдаваться в анализ сходства и различий в трактовке смешанных вычислений у автора и аналогичных конструкций, предложенных в только что упомянутых работах. Достаточно лишь отметить, что, с одной стороны, идея смешанных вычислений и понятие остаточной программы объективно существуют уже свыше десяти лет, а, с другой стороны эти концепции, впервые выдвинутые Л.Ломбарди, поддержанные его последователями и существенно продвинутые исследователями, программирующими на Лиспе и Рефале, до

241-163

последнего времени не оказали подобающего влияния на теорию и практику трансляции.

Автору приятно было приобщиться к кругу тех, кто додумался до принципа частичных вычислений, однако главным моментом своего исследования он хотел бы сделать фундаментализацию этого принципа, его выражение и обоснование в понятиях, общих для разных алгоритмических языков.

Мы привычно говорим о синтаксисе и семантике языков программирования. В последние 15 лет фундаментальная теория синтаксического анализа формальных языков получила большое развитие и предоставила практике и необходимые понятия и надлежащую технику. Собственно же трансляция и поныне остается гораздо менее изученной частью. В то же время болезненный опыт реализации таких сложных языков как ПЛ/I и Алгол 68 делает задачи оптимизации и генерации весьма актуальными. С этой точки зрения смешанные вычисления выглядят весьма многообещающ средством единого описания и обоснования многих сущностей и реальностей в этих разделах трансляции.

241-164

Литература

[1] F.G.Fagan. On interpreter-oriented definitions of programming languages. Comp. Journ., 19, No. 2, 1976, 151-155.

[2] A.P.Ershov, V.B.Itkin. Correctness of mixed computation in Algol-like programs. A paper to be presented to the 6th conference on mathematical foundations of computer science. Tatranska Lomnica, CSSR, September 1-5, 1977. To be published in LNCS, Springer-Verlag, Heidelberg, 1977.

[3] A.P.Ershov, V.V.Grushetsky. An implementation-oriented method for describing algorithmic languages. A paper to be presented at the IFIP 77 Congress, 8-12 August 1977, Toronto, Canada.

[4] F.E.Allen, J.Cocke. A catalogue of optimizing transformtions. IBM Research RC 3548, Yorktown Heights, 1971.

[5] И.В.Поттосин. Глобальная оптимизация, практический подход. Труды всесоюзного симпозиума по методам реализации новых алгоритмических языков. Часть I. ВЦ СО АН СССР, Новосибирск, 1975, 113-128.

[6] А.П.Ершов (ред.) АЛЬФА - система автоматизации программирования. Наука, Новосибирск, 1967.

[7] А.П.Ершов. Основные принципы построения программирующей программы Института математики СО АН СССР. Сибирский математ. журнал, 2, № 6, 1961.

[8] А.П.Ершов. Универсальный программирующий процессор. В сб. "Проблемы прикладной математики и механики". Наука, М., 1971, 103-116.

[9] A.P.Ershov. The British lectures, BCS, London (to be published).

[10] А.П.Ершов. Об одном принципе системного программирования. ДАН СССР, 233, № 2, 1977, 272-275.

[11] L.A.Lombardi. Incremental data assimilation in open-ended man-computer system (Research summary). Memorandum MAC-M-104(S63). October 1, 1963, PROJECT MAC, MIT, Cambridge.

[12] L.A.Lombardi. Incremental computation. Advances in computers, v. 8, 1967.

241-165

[13] Г.Х.Бабич. Алгоритмический язык ДекАС для решения задач с неполной информацией и алгоритмы его интерпретации. Кибернетика, № 2, 1974, 61-71.

[14] Г.Х.Бабич, Л.Ф.Штернберг, Т.И.Юганова. Алгоритмический язык Инкол для выполнения вычислений с неполной информацией. Программирование, № 4, 1976, 24-32.

[15] В.Ф.Турчин. Метаязык для формального описания алгоритмических языков. Цифровая вычислительная техника и программирование. Советское Радио, М., 1966, 116-124.

[16] В.Ф.Турчин. Эквивалентные преобразования программ на РЕФАЛе. Автоматизированная система управления строительством. Труды института, вып. 6, ЦНИПИАСС, М., 1974, 36-68.

[17] L.A.Lombardi, B.Raphael. LISP as the language for an incremental computer. The programming language LISP: its operation and applications. Information International, Inc., Cambridge, March 1964, 204-219.

[18] L.Beckman, et al. A partial evaluator and its use as a programming tool. Datalogilaboratoriet, memo 74/34, Uppsala University, Uppsala, 1974 (also: Artificial Intelligence, vol. 7, 1976, 319-357).

[19] E.Sandewall. Ideas about management of LISP data bases. Advance papers of the 4th IJCAI, Tbilisi, 3-8 September 1975, vol. 2, 585-592.

[20] P.Henderson, J.H.Morris, Jr. A lazy evaluator. Techn. Report No. 75, January 1976, Computing Laboratory, The University of Newcastle upon Tyne (also: 3rd ACM Symposium on principles of programming languages, January 19-21, 1976, Atlanta).

[21] M.Elson, S.T.Rake. Code-generation technique for large-language compilers. IBM Systems Journal, No. 3, 1970, 166-188.

[22] W.Harrison. A new strategy for code generation - the General Purpose Optimizing Compiler. IBM T.J.Watson Research Center, Yorktown Heights, 1976 (also: 4th ACM Symposium on principles of programming languages, 1977).

241-166

[23] A.P.Ershov. Problems in many-language programming systems. Language hierarchies and interfaces. LNCS, vol 46, Springer-Verlag, Heidelberg, 1976, 358-428.

24 А.П.Ершов. Проектные характеристики многоязыковой системы программирования. Кибернетика, № 4, 1975, II-27.

[25] Y.Futamura. Partial evaluation of computer programs: an approach to a compiler-compiler. Journ. of Inst. of electronics and communication engineers, 1971.