ИТЭФ —57

ИНСТИТУТ ТЕОРЕТИЧЕСКОЙ И
ЭКСПЕРИМЕНТАЛЬНОЙ ФИЗИКИ

Р.Ф. ГУРИН

МЕТОД ОПТИМИЗАЦИИ ПРОГРАММ ОБРАБОТКИ СПИСКОВ

МОСКВА 1984

УДК 681.142.5:539.1.07 М-16

Предлагается оптимизировать программы обработки списков выбором индивидуального представления каждого возникающего в программе значения и настройкой каждого оператора в программе на специфическое представление обрабатываемых им данных. Основой для этого служат оценки областей изменения значений различных выражений в программе. Предлагается метод для получения таких оценок.

§ I. ВВЕДЕНИЕ

В существующих системах обработки списков представление списков в памяти либо фиксируется априори, как, например, в лиспе /I/, либо выбор этого представления поручается программисту. В данной работе предлагается возложить этот выбор на компилятор.

В § 2 описывается простой язык обработки списков, в дальнейшем называемый базовым. Предполагается, что этот язык или некоторое его расширение будет использоваться для трансляции и а него программ на каком-либо реально существующем языке обработки списков. Полученные промежуточные программы оптимизируются, а затем исполняются.

В § 3 предлагается метод анализа программ на базовом языке. Этот метод позволяет оценить множества возможных значений переменных в различных точках программы, множества возможных значений аргументов и результатов имеющихся в программе функций.

В § 4 описываются возможные оптимизации программы и представления обрабатываемых ею данных на основе полученных для нее оценок.

Наконец, в § 5 обсуждаются расширения базового языка, необходимые для его реального использования.

§ 2. БАЗОВЫЙ ЯЗЫК

Обрабатываемыми объектами в программах на базовом языке являются атомы и списки. Набор атомов фиксировать пока не будем. Список определим как линейную последовательность (возможно, пустую) атомов и списков. Элементы списков будем называть термами, чтобы не перегружать слово "элемент". Переменные в программах на базовом языке не имеют фиксированного типа, их значениями могут быть как атомы, так и списки.

Синтаксис языка опишем в расширенной БНФ:

<программа> ::= <блок> .

<блок> ::= begin
[var <переменная> {, <переменная>};]
{<описание функции> ;}
{[<метка> :] <оператор> ;}
end .

<описание функции> ::= function <имя функции>
([<форм. параметр> {, <форм. параметр>}])
<блок> .

<оператор> ::= <оператор присваивания> |
<оператор перехода> |
<оператор возврата из функции> |
<условный оператор> .

<оператор присваивания> ::=
<переменная> := <выражение> .

<оператор перехода> ::= goto <метка> .

<оператор возврата из функции> ::= return <выражение> .

<условный оператор> ::= if <выражение>
then <оператор> .

<форм. параметр> ::= <переменная> .

<выражение> ::= <атом> |
<переменная> |
<вызов функции> .

<вызов функции> ::=
<имя функции> ([<аргумент> {, <аргумент>}]) .

<аргумент> ::= <выражение> .

В приведенной записи угловые, квадратные и фигурные скобки - метасимволы с общепринятым смыслом. Понятие, заключенное в квадратные скобки, может быть опущено, а понятие, заключенное в фигурные скобки, может быть повторено произвольное число раз, в том числе нуль раз. Конкретный вид атомов, меток, переменных оставлен неопределенным. Для целей данной работы он несущественен.

Опишем имеющиеся в языке встроенные функции:

CAR(x). Аргумент - список. Результат - самый левый терм этого списка.

CDR(x). Аргумент - список. Результат - этот список без самого левого терма.

NIL(). Аргументов нет. Результат - пустой список.

APPEND(x,y). Оба аргумента должны быть списками. Результат - конкатенация этих списков.

TERM(x). Аргумент - произвольное значение. Результат - список, единственным термом которого является это значение.

Кроме того, имеется ряд встроенных функций-предикатов. Их значение - всегда либо специальный атом И, обозначающий

истину, либо специальный атом Л, обозначающий ложь. В качестве примера приведем функцию АТОМ, проверяющую является ли ее аргумент атомом, и функцию ПУСТО, проверяющую является ли ее аргумент пустым списком.

Завершим описание базового языка несколькими отдельными замечаниями. При вызове любой функции аргументы передаются по значению. Любая функция вырабатывает значение, выход из функции возможен только по оператору return . Значением выражения после if в условном операторе должно быть И или Л. Использование APPEND вместо более привычного CONS вызвано желанием оставить возможность эффективной реализации APPEND, избежать копирования первого аргумента, требующегося при реализации APPEND через CONS .

§ 3. АНАЛИЗ ПРОГРАММ НА БАЗОВОМ ЯЗЫКЕ

Основная цель этого анализа - получить для различных выражений в программе оценки множеств их возможных значений. Методы подобного анализа предлагаются в работах /2,3/. К сожалению, описанные в этих работах методы мало пригодны для программ, работающих с данными, чья структура определяется рекурсивно. В частности, они мало пригодны для программ обработки списков. Поэтому предложим свой метод.

Перенумеруем все имеющиеся в тексте программы выражения, включив в их число формальные параметры функций и переменные в правых частях операторов присваивания. Выражение с номером k будем обозначать через E^k. Для каждого

выражения будем оценивать множество его возможных значений. Представление оценочного множества возможных для E^K значений-атомов будем обозначать через A^K, а значений-списков - через C^K. Сами эти оценочные множества будем обозначать через знач A^K и знач C^K

A^K возьмем из работы /3/. То есть: на множестве всех атомов введем решетку типов, в которой все цепи конечны (см. работу /3/), и в качестве A^K будем использовать элементы этой решетки. Если T - элемент решетки, то через знач (T) будем обозначать множество атомов, имеющих тип T. Частичный порядок на решетке задается так:

$$T^a \leqslant T^B \iff \text{знач}(T^a) \subseteq \text{знач}(T^B)$$

На множестве элементов решетки должна быть определена операция $\vee$ (наименьшая верхняя грань) со свойствами:

$$x \vee y \geqslant x \ , \quad x \vee y \geqslant y$$

На множестве атомов должна быть определена функция "тип", такая, что тип (a) - элемент решетки типов, и знач (тип (a)) $\ni$ a . Решетка типов должна включать специальный элемент $\perp$ (пустой тип) со свойством знач ($\perp$) = $\varnothing$, и специальный элемент $\top$ (универсальный тип) со свойством знач($\top$) = {множество всех атомов} .

Опишем теперь структуру C^K. Каждое C^K есть кортеж из четырех элементов - матрицы M^K, двух векторов $ВЛ^K$ и $ВП^K$ и скаляра $П^K$. Размер векторов - N , а матрицы - $N \times N$, где N - число выражений в программе. Значение скаляров и элементов векторов и матриц - 0 либо 1. Скаляр $П^K$ показывает, может ли значение E^K быть пустым списком. Единица означает "да", нуль - "нет". Матрица M^K показывает, из каких термов состоят возможные значения-списки E^K и каков

порядок этих термов. Единица в позиции a,в матрицы M^K означает, что, во-первых, выражения E^a и E^B имеют вид ТЕРМ (выражение), а во-вторых, их значения могут соседствовать на нулевом уровне списочной структуры в значениях E^K. Вектора $ВЛ^K$ и $ВП^K$ показывают, какие термы могут быть крайними термами значений E^K. Единица в позиции a вектора $ВЛ^K$ означает, что, во-первых, выражение E^a имеет вид ТЕРМ выражение, а во-вторых, значение E^K может начинаться со значения E^a. Единица в позиции a вектора $ВП^K$ означает, что, во-первых, выражение E^a имеет вид ТЕРМ выражение, а во-вторых, значение E^K может оканчиваться значением E^a.

На множестве этих скаляров, векторов и матриц введем обычным образом операции сложения и умножения, с одной лишь поправкой: I + I = I. Определим операцию сложения также на множестве кортежей C^K. Будем складывать их поэлементно. Упорядочим множество {0,I}, задав 0 < I, и введем частичный порядок на множествах векторов, матриц и кортежей. Для двух векторов (матриц, кортежей) X^a и X^B выполнено $X^a \leq X^B$, если и только если каждый элемент X^a меньше соответствующего элемента X^B или равен ему.

Нам потребуются еще две матрицы того же размера, что и M^K. Первая из них - матрица ТРМ. $ТРМ_{a,B}$ = I означает, что выражение с номером a есть ТЕРМ (E^B). Вторая - матрица ИСП. $ИСП_{a,B}$ = I означает, что значение E^B может быть использовано в качестве значения E^a. Это возможно, например, при наличии в программе присваивания $E^a := E^B$.

Теперь приведем ряд формул.

Пусть даны C^a и C^B. Пусть E^C есть APPEND(E^a,E^b). Тогда все возможные значения E^C можно описать с помощью

$$C^C = \langle M^a \cdot M^B + B\Pi^a \cdot (B\Pi^B)^*,\ B\Pi^a + \Pi^a \cdot B\Pi^B,\ B\Pi^B + \Pi^B \cdot B\Pi^a,\ \Pi^a \cdot \Pi^B \rangle \overset{def}{=} append(C^a, C^B)$$

Поясним эту формулу. Первый элемент кортежа C^C показывает, что в значении E^C могут соседствовать те же термы, что и в значении E^a, те же термы, что и в значении E^B, а также крайние правые термы из значений E^a с крайними левыми термами из значений E^B. Надстрочная звездочка означает транспонирование. $B\Pi^B$ - вектор-столбец. Соответственно $(B\Pi^B)^*$ - вектор-строка. Второй элемент кортежа показывает, что значения E^C могут начинаться с тех же термов, что и значения E^a, а также, при пустом значении E^a, с тех же термов, что и значения E^B. Третий элемент аналогичен второму и показывает, на какие термы может оканчиваться значение E^C. Наконец, четвертый элемент показывает, что E^C может принимать пустые значения только если их могут принимать и E^a и E^B.

Пусть дано C^a и пусть E^B есть $CDR(E^a)$. Тогда все возможные значения E^B можно описать с помощью $C^B = \langle M^a,\ (B\Pi^a)^* \cdot M^a,\ B\Pi^a,\ (B\Pi^a, B\Pi^a) \rangle \overset{def}{=} cdr(C^a)$. Дадим пояснение к четвертому элементу кортежа C^B. Запись $(B\Pi^a, B\Pi^a)$ означает скалярное произведение векторов $B\Pi^a$ и $B\Pi^a$. Это скалярное произведение может равняться единице только если значение E^a может начинаться и заканчиваться одним и тем же термом, т.е. состоять из одного терма. Только в этом случае значение $CDR(E^a)$ может быть пустым.

Пусть теперь E^B есть $CAR(E^a)$. Если значение E^a может начинаться с ТЕРМ (E^C), то значением $CAR(E^a)$ может быть значение E^C. Проще всего отразить этот факт,

внеся единицу в соответствующую позицию матрицы ИСП. Перебирая все термы, с которых может начинаться значение E^a, получим $ИСП_{в,*} \geq (ВЛ^a)^* * ТРМ$, где через $ИСП_{в,*}$ обозначена в-я строка матрицы ИСП.

Рассмотрим теперь элементы матрицы ИСП. $ИСП_{а,в} = I$ означает, что некоторые значения $E^в$ должны учитываться также как значения E^a. Это легко обеспечить, потребовав $знач(A^a) \geq знач(A^в)$, $знач(C^a) \geq знач(C^в)$. Это, в свою очередь, легко обеспечить, потребовав $A^a \geq A^в$, $C^a \geq C^в$.

Вычислим по программе матрицу ТРМ и выпишем соотношения, которым должны удовлетворять матрица ИСП, кортежи $C^к$ и типы $A^к$.

Рассмотрим вхождения в текст программы различных переменных. При нумерации выражений каждое из них получит уникальный номер к. С каждым вхождением к переменной х свяжем список вхождений $к_1, к_2, \ldots к_i$ этой переменной, в которых может быть использовано значение, полученное во вхождении к. Такие списки могут быть получены при помощи хорошо известных алгоритмов анализа потока данных, описанных, например, в /4,5/. Попутно свяжем с каждой левой частью оператора присваивания его правую часть, с каждым формальным параметром - фактические аргументы, с каждым выражением вида $f(\ldots)$ - выражения из операторов return в теле функции f. Если с выражением $E^к$ связаны выражения $E^{к_1}$, $E^{к_2}$, ... $E^{к_i}$, потребуем

$$ИСП_{к_1,к}, ИСП_{к_2,к}, \ldots ИСП_{к_i,к} = I \qquad (I)$$

- для каждого $E^в \in ТЕРМ(E^a)$ потребуем

$$ВЛ^в_в = ВП^в_в = I \qquad (2)$$

- для каждого $E^a \equiv NIL()$ потребуем

$$П^a = I \qquad (3)$$

- для каждого $E^a \equiv f(\dots)$, где f - предикат, потребуем $A^a = тип(И) \vee тип(Л)$ (4)

- для каждого E^a, являющегося атомом, потребуем

$$A^a = тип(E^a) \qquad (5)$$

- для каждого $E^c \equiv APPEND(E^a, E^b)$ потребуем

$$C^c = append(C^a, C^b) \qquad (6)$$

- для каждого $E^B \equiv CDR(E^a)$ потребуем

$$C^B = cdr(C^a) \qquad (7)$$

- для каждого $E^B \equiv CAR(E^a)$ потребуем

$$ИСП_{B,*} \geqslant (ВЛ^a)^* * ТРМ \qquad (8)$$

- потребуем

$$ИСП_{a,B} = I \Rightarrow (C^a \geqslant C^B) \,\&\, (A^a \geqslant A^B) \qquad (9).$$

Сделаем утверждение без доказательства: если набор C^K, A^K и матрица ИСП удовлетворяют условиям I-9, то любое значение любого выражения E^K в любом прогоне программы входит в $знач(C^K) \cup знач(A^K)$. Искать этот набор будем так: сначала установим $ИСП_{a,B}$, $М^a_{B,c}$, $ВЛ^a_B$, $ВП^a_B$, $П^a$ в соответствии с равенствами I-3, т.е. элементы, перечисленные в равенствах I-3, устанавливаем в I, остальные - в 0. Затем устанавливаем A^K в соответствии с равенствами 4-5. Те A^K, которые не перечислены в этих равенствах, устанавливаем в $\perp$. Далее повторяем в цикле следующие действия:

- если не выполнено какое-то из равенств 6 или 7, то выполняем его, присваивая его левой части значение правой;

- если не выполнено какое-то из неравенств 8, то выполняем его, заменяя для соответствующих a и в $ИСП_{B,*}$

на $ИСП_{B,\alpha}$ + (ВЛ^{a})→ТРМ;

- если не выполнено какое-то из условий 9, то выполняем его, заменяя для соответствующих a и в C^a на $C^a + C^B$ и A^a на $A^a \vee A^B$.

Цикл повторяется до тех пор, пока все условия I-9 не будут выполнены.

Сделаем опять несколько утверждений без доказательства:

- алгоритм завершается

- он дает минимальный набор ИСП, C^K, A^K, удовлетворяющий условиям I-9. Т.е. для любого другого набора ИСП', $(C^K)'$, $(A^K)'$, удовлетворяющего этим условиям, будем иметь ИСП' ⩾ ИСП, $(C^K)' \geqslant C^K$, $(A^K)' \geqslant A^K$.

Для практической реализации алгоритма важны два обстоятельства. Во-первых, для всякой реальной программы матрицы M^K и вектора $ВЛ^B$, $ВП^K$ будут сильно разреженными. Это делает алгоритм приемлемым по памяти. Во-вторых, можно организовать выполнение алгоритма так, чтобы избежать многих вычислений. Например, замену C^a на $C^a + C^B$, на первый взгляд, требующую $O(N^2)$ операций, можно выполнять гораздо быстрее, если учитывать только те элементы C^B, которые изменились со времени последней такой замены. Теоретически наихудшее время работы алгоритма $O(N^4)$, однако, в экспериментах время работы зависит от N приблизительно квадратично.

§ 4. ОПТИМИЗАЦИЯ ПРОГРАММ НА БАЗОВОМ ЯЗЫКЕ

На основе собранной информации можно доказать ряд утверждений о значениях различных выражений в программе:

- если $A^K = \perp$, то заведомо АТОМ (E^K) = Л (1);

- если $П^K = 0$, то заведомо ПУСТО (E^K) = Л (2);

- если в $БЛ^K$ или $ВП^K$ нет ни одного ненулевого элемента, то значением E^K не может быть непустой список, т.е. выполнено (АТОМ (E^K) = И) $\vee$ (ПУСТО (E^K) = И)(3);

- если одновременно выполнено 1 и 3 , то заведомо ПУСТО (E^K) = И ;

- если одновременно выполнено 2 и 3 , то заведомо АТОМ (E^K) = И .

Такие утверждения позволяют, с одной стороны, заменить вызовы некоторых предикатов в программе константами, с другой стороны, для некоторых вызовов функций CAR и CDR отказаться от проверки корректности аргумента.

Рассмотрим теперь представление каждого терма в памяти. Будем считать, что в отличие от обычно используемого представления списков, когда каждый терм занимает отдельный блок квант, ячейку памяти, возможно соединение нескольких термов в один блок. При этом не расходуется память на поля связи между термами. Для каждого терма будем выбирать следующие параметры:

- длину поля дескриптора dlen . Поле дескриптора содержит информацию о том, является данный терм атомом или списком и, возможно, другую информацию, описывающую терм, например, является ли он числом и т.п.

- длину информационного поля ilen . Содержимое этого поля зависит от типа терма. Это может быть указатель на список, число или что-то иное;

- позицию pos этих полей в блоке памяти, содержащем данный терм;

- индикаторы ℓ и τ, показывающие, является ли данный терм самым левым и/или самым правым в своем блоке.

Потребуем, чтобы для выражений $CAR(E^a)$ и $CDR(E^a)$ представление самого левого терма в значении E^a было известно статически. Это позволит вычислять эти функции столь же быстро, как, например, в лиспе, не загромождая программу динамическим анализом информации о представлении данных. Отсюда вытекает требование: если в программе имеется выражение $CAR(E^a)$ или $CDR(E^a)$, то все выражения вида ТЕРМ (E^x), номера которых перечислены в $ВЛ^a$, должны порождать одинаково представленные термы. Собранные вместе, такие требования разбивают множество выражений вида ТЕРМ (E^k) на классы эквивалентности.

Рассмотрим один из этих классов. Пусть он состоит из выражений ТЕРМ (E^{k_1}), ТЕРМ (E^{k_2}), ... ТЕРМ (E^{k_i}). Если все E^{k_j} обладают каким-либо общим свойством, например, для них всех АТОМ (E^{k_j}) = И или все они являются числами, то в дескрипторах термов, порождаемых выражениями ТЕРМ (E^{k_j}), можно не указывать информацию об обладании этим свойством. Это может позволить уменьшить длину поля дескриптора или даже довести ее до нуля. Аналогичным образом можно уменьшить длину информационного поля этих термов. Например, если заведомо известно, что все E^{k_j} принимают значение И или Л, то под информационное поле можно отвести всего один бит.

Параметры pos, ℓ, τ представлений термов выбираются на основе информации о возможном соседстве термов, т.е. на основе матрицы $M = \sum_{k=1}^{K} M^k$. При этом должны выполняться

следующие соотношения

$$(M_{a,b} = 1) \Rightarrow (r(E^a) \Leftrightarrow l(E^b)),$$

$$(M_{a,b} < 1 \;\&\; \neg r(E^a)) \Rightarrow$$
$$pos(E^b) = pos(E^a) + dlen(E^a) + ilen(E^a).$$

§ 5. ЗАКЛЮЧЕНИЕ

Выбор различных представлений для различных термов в программе позволяет уменьшить расход памяти на представление данных. В то же время, приведенный метод такого выбора не приводит к появлению в процессе работы объектной программы дополнительных накладных расходов на анализ представления данных. Наоборот, он позволяет уменьшить время работы программы за счет удаления из нее некоторых проверок, определяя их результат статически.

Для практического применения метода необходимо:

- расширить базовый язык, введя в него средства раздельной компиляции, новые встроенные функции, какой-то аналог функции EVAL лиспа. При этом надо соответствующим образом модифицировать метод анализа программ.

- рассмотреть вопросы, связанные с динамическим распределением памяти и сборкой мусора.

ЛИТЕРАТУРА

1. M c C a r t h y T. Recursive Functions of Symbolic Expressions and their Computation by Machine. - Communications of the ACM, 1960, v.3, p.184-195.

2. T e n e n b a u m A. Rovised and Extended Algorithms for Deducing the Types of Objects Appearing in SETL Programs. - SETL Newsletter, Oct.1973, No.118, N.Y.Univ.

3. Kan p l a n M.A., U l l m a n T.D., A Scheme for the Automatic Inference of Variable Types. - Journal of the ACM, 1980, v.27, no.1, p.128-145.

4. A l l e n F.E., C o c k e J. A Program Data Flow Analysis Procedure. - Communications of the ACM, 1976, v.19, no.3, p.137-147.

5. B a r t h J.M. A Practical Interprocedural Data Flow Analysis Algorithm, - Communications of the ACM, 1978, v.21, no.9, p.724-736.

Гурин Р.Ф.
Метод оптимизации программ обработки списков

Редактор И.Н.Ломакина Корректор О.Ю.Ольховникова
Работа поступила в ОНТИ 13.04.84

Подписано к печати 24.03.84 Т09880 Формат 60х90 1/16
Офсетн.печ. Усл.-печ.л.1,0. Уч.-изд.л.0,7. Тираж 130 экз.
Заказ 57 Индекс 3624 Цена 10 коп.

Отпечатано в ИТЭФ, 117259, Москва, Б.Черемушкинская, 25

10 коп ИНДЕКС 3624

М.,ПРЕПРИНТ ИТЭФ, 1984, № 57, с.1–15