Э.Ю. Петров

СКЛЕИВАНИЕ ПЕРЕМЕННЫХ В ПРЕДИКАТНОЙ ПРОГРАММЕ

Склеивание переменных есть замена в программе нескольких переменных одной. Например, склеивание переменных x, y и z представляет систематическую замену всех вхождений имен y и z на имя x. Разумеется, программа после склеивания должна быть эквивалентна предыдущей.

Склеивание переменных рассматривалось ранее в рамках задачи экономии памяти в классических работах А.П. Ершова, С.С. Лаврова, В.В. Мартынюка. Склеивание переменных определялось как выбор переобозначения аргументов и результатов из заданного множества корректных переобозначений, которое позволяет в наибольшей степени уменьшить объем необходимой памяти [3]. В оптимизирующей трансляции императивных программ склеивание переменных обычно реализуется на основе анализа локальных свойств программы [4, 5].

Технология предикатного программирования предполагает написание программы на языке предикатного программирования Р [2], применение к ней набора оптимизирующих трансформаций с получением программы на императивном расширении языка Р и конвертацию на любой из императивных языков: С, С++ и др. Склеивание переменных реализуется на первом этапе трансформации предикатной программы в императивную.

Предикатная программа представляет собой набор рекурсивных определений предикатов. В определении предиката склеиванию подлежат совместимые по типу локальные и результирующие переменные с исходными переменными. Целью склеивания является оптимизация, прежде всего, по времени исполнения. Значительный эффект достигается при склеивании структурных переменных — массивов и последовательностей.

Задача склеивания переменных в данной постановке рассматривается впервые. Она вряд ли может стать актуальной для оптимизации функциональных программ**:** в определении функции нет результирующих переменных, и там можно было бы рассматривать лишь склеивание локальных и исходных переменных. Эта задача не возникает для императивных программ, поскольку практически все рассматриваемые здесь склеивания обычно реализуются программистом в императивной программе.

1. МОДЕЛЬ ПРОГРАММЫ

Задача склеивания переменных формулируется для модели программы, соответствующей языку предикатного программирования Р [2].

1.1. Определение предиката

Предикат есть условие, определяющее зависимость между значениями переменных. Пусть дан предикат A(x, z), где A — обозначает имя предиката, x — набор **входных** (используемых) переменных $x_1,x_2,...,x_n$ ($n \geq 0$), а z — набор **результирующих** (определяемых) переменных $z_1,z_2,...,z_m$ ($m>0$); наборы x и z не пересекаются. Будем использовать запись A(**x:** z), где "**:**" разделяет входные и результирующие переменные.

Определение предиката имеет следующую форму:

<имя предиката> **(**<входные параметры> **:** < результирующие параметры>**)**
{ <тело предиката> **}**

Вычисление предиката реализует исполнение тела предиката для получения значений результирующих параметров по значениям входных.

Описание каждого параметра предиката имеет вид: <тип> <имя параметра>. Тип может быть примитивным, подмножеством типа или структурным. Примитивные типы это **nat**, **int**, **real** или **bool**. Структурными типами являются: массив, кортеж, объединение, последовательность и множество.

Частным случаем подмножества типа является диапазон, задаваемый следующим образом:

<выражение>..<выражение>

Например, 1..n+1 обозначает диапазон натуральных чисел, где переменная n является параметром типа.

Описание типа "последовательность" имеет вид:

seq <тип элементов последовательности>

Описание типа "массив" имеет вид:

array < тип индекса> **of** < тип элементов массива>

Тип индекса обычно является диапазоном.

Предикат вида A(x**:** y) соответствует функции. Определим предикат более общего вида в форме **гиперфункции.** Допустим, наряду с предикатом A(x**:** z) имеется предикат B(x**:** y). Списки переменных z и y не содержат общих переменных. Области определения функций A и B, соответственно, A_x и

B_x, не пересекаются. Гиперфункция C(x**:** z | y) с двумя **ветвями** есть предикат, определяемый формулой:

$$(x \in A_x \Rightarrow A(x{:}\ z)) \ \&\ (x \in B_x \Rightarrow B(x{:}\ y))$$

Гиперфункция C(x**:** z | y) определена на $A_x \cup B_x$ и совпадает с функцией A на A_x и с функцией B на B_x. Вычисление гиперфункции завершается на одной из двух ветвей с получением значений результирующих переменных завершившейся ветви; переменные другой ветви не вычисляются. Гиперфункция C реализует ветвление — она совмещает в себе преобразователь и распознаватель. Ветвь гиперфункции может быть **пустой**, т.е. не содержать результирующих переменных. По пустой ветви гиперфункция является только распознавателем.

В общем случае, гиперфункция может содержать произвольное количество ветвей.

Исполнение гиперфункции завершается некоторой ветвью с вычислением значений набора результирующих параметров ветви. Для указания того, какой ветвью завершилось вычисление гиперфункции, используется завершитель ветви: **#**<номер ветви>.

1.2. Операторы

Первичными операторами являются: предикат равенства, вызов предиката и оператор разложения.

Предикат равенства имеет вид:

<переменная> **=** <выражение>

<переменная> это либо результирующий параметр определения предиката, либо локальная переменная, декларированная выше в программе описанием вида**:**

< тип> <имя переменной>**.**

Вызов предиката имеет следующий вид:

<имя предиката>([<аргументы>]**:**<результаты>)

Вычисление вызова предиката производится следующим образом. Сначала независимо (параллельно) вычисляются все выражения, находящиеся в позиции аргументов предиката. Полученные значения присваиваются во входные параметры определения вызываемого предиката. Далее выполняется тело предиката, в котором происходит вычисление результирующих параметров. Значения результирующих параметров присваиваются переменным, являющимися результирующими фактическими параметрами в вызове предиката.

На базе примитивных предикатов строятся следующие операторные композиции: суперпозиция (блок), условный оператор, оператор расщепления, параллельный оператор и конструктор массива.

Пусть имеются два оператора: A(x: y) и B(u: z), причем наборы x и z не содержат общих переменных. Оператор A(x: y); B(u: z) является **суперпозицией**, если наборы y и u содержат общие переменные. **Оператор** A(x: y) || B(u: z) является **параллельным оператором**, если в наборах u и y нет общих переменных.

Суперпозиция нескольких операторов образует **блок.** Перед любым оператором блока могут помещаться описания локальных переменных. Блок обычно обрамляется фигурными скобками.

Условный оператор имеет следующий вид:

if <выражение> **then** <сегмент>
else <сегмент>
end

Сегментом является блок и следующий за ним необязательный завершитель ветви. Фигурные скобки блока, составляющего сегмент, можно опускать. Если сегменты условного оператора заканчиваются одинаковыми завершителями, то каждый сегмент должен определять одинаковый набор результирующих переменных.

Оператор расщепления имеет следующий вид:

split <вызов предиката-гиперфункции>
do <первая альтернатива расщепления>
do <вторая альтернатива расщепления>
end

Каждая альтернатива расщепления есть <сегмент>. Исполнение оператора расщепления начинается с вызова предиката-гиперфункции. Если исполнение вызова гиперфункции завершается первой ветвью, то далее исполняется первая альтернатива расщепления, иначе — вторая. На первой ветви расщепления используются переменные, вырабатываемые первой ветвью вызова гиперфункции; аналогично — для второй ветви расщепления.

Определение оператора расщепления обобщается на произвольное число ветвей предиката-гиперфункции.

Для присваивания значений переменным типа «массив» определен оператор — конструктор массива:

forAll <простая переменная> = <диапазон> **do** <оператор> **end**

Для каждого значения <простой переменной>, принадлежащего <диапазону>, независимо (параллельно) исполняется <оператор>, вычисляющий элемент массива. Например, в следующем фрагменте конструируется массив, элементы которого равны 1:

```
int n;
....
array 1..n of int arr;
forAll i = 1..n do arr[i] = 1 end;
```

Оператор разложения предназначен для определения компонент объекта структурного типа. Например, разложение последовательности s->(| e, r) представляет следующее утверждение: последовательность s либо пуста, и тогда реализуется первая ветвь гиперфункции, либо s состоит из элемента e и подпоследовательности r.

Пример 1.

```
Length(seq int s: int len) {
    Length2(s, 0: len)
}

Length2(seq int s, int acc : int len) {
    seq int r;
    int e;
    split s->( | e, r)
    do  len = acc;
    do  Length2 (r, acc+1: len);
    end;
}
```

В примере 1 предикат Length определяет длину последовательности s. Используется вспомогательный предикат Length2, определяемый условием:

Length2 (s, acc) = Length(s)+acc

1.3. Дополнительные определения

Для условного оператора и оператора расщепления будем использовать общее понятие — **оператор ветвления**, состоящий из **ветвей** и **заголовка** — части, предшествующей ветвлению.

Для некоторого оператора G, находящегося в теле некоторого определения предиката, введем обозначения, используемые для описания алгоритма склеивания.

Назовем **аргументами** G параметры предиката и локалы, использующиеся в G. Аргументы G обозначим **Args(G)**. Переменные из Args(G), которые не используются при продолжении исполнения после исполнения оператора G, обозначим **A(G)**. Склеивание переменной из A(G) с другими переменными является допустимой трансформаций, сохраняющей эквивалентность программы по результатам исполнения.

Множество переменных, определяемых (присваиваемых) в операторе G и либо использующихся при дальнейшем исполнении определения предиката после исполнения G, либо являющимися результирующими параметрами предиката, будем называть результатами оператора G и обозначим **R(G)**.

Переменные, определяемые в G и неиспользуемые далее при исполнении тела предиката после G, назовем **собственными локалами** G.

2. ПОСТАНОВКА ЗАДАЧИ

Склеивание переменных есть преобразование предикатной программы, при котором несколько переменных, совместимых по типу, заменяются одной. Например, при склеивании переменных c и d оператор c=d+1 будет преобразован в оператор присваивания c:=c+1, а оператор a = b при склеивании a и b превратится в тождественный предикат a = a, удаляемый из императивной программы. В отличие от задачи экономии памяти [3], склеиванию подлежат только те переменные, между которыми имеется информационная связь.

Типы считаются совместимыми, если они либо тождественны, либо являются конкретизациями одного общего типа с разными значениями параметров типа. Понятие совместимости типов используется здесь в более узком смысле, чем в описании языка Р [2].

Склеивание переменных реализуется на первом этапе в процессе получения эффективной императивной программы и предваряет все остальные трансформации: преобразование хвостовой рекурсии в цикл, подстановку тела предиката на место вызова и конкретизацию операций с переменными структурных типов.

В данной работе рассматривается алгоритм склеивания переменных в пределах одного определения предиката. Правилами языка Р предусматривается возможность склеивания аргументов и результатов для произвольного определяемого предиката. При наличии такого склеивания функциями алгоритма является проверка корректности такого склеивания. Полный ал-

горитм, реализующий также склеивание входных и результирующих параметров предикатов, предполагается разработать в дальнейшем.

В работе не рассматривается склеивание переменных структурных типов разной длины, за исключением склеивания, подразумеваемого в операторе разложения последовательности. Также не рассматривается склеивание переменных типа «массив» в операторе forAll. Например, в следующем фрагменте можно было бы заменить переменную b на a при условии, что в дальнейшем вычислении a не используется.

Пример 2.

```
int n;
array 1..n of int a;
array 1..n+1 of int b;
...
forAll i = 1..n+1 do
        if i <5 then b[i] = a[i] elsif i =5 then b[5] = 0 else b[i] = a[i-1] end;
end;
```

3. ОБЩИЙ АНАЛИЗ И ОПИСАНИЕ АЛГОРИТМА

Обозначим через **<x: y> команду склеивания**, определенную по отношению к некоторому оператору G, где x — аргумент из A(G), а y — результирующая переменная в G. Выполнение этой команды заменяет y на x в G, а также во всех операторах, исполняющихся после завершения исполнения G.

Определим алгоритм построения совокупности команд склеивания для некоторого определения предиката. Применение этой совокупности команд к определению предиката реализует полный набор склеиваний в определении предиката.

Для каждой результирующей переменной $y_k \in R(G)$ $(k = 1,...,m)$ строится **кандидатная пара** $<A_k(G): y_k>$, где: y_k информационно зависима от всех переменных из $A_k(G)$, причем y_k и переменные из $A_k(G)$ — совместимы по типу. $A_k(G)$ называется **множеством кандидатов** для склеивания с y_k.

Перечислим свойства склеивания переменных для произвольного оператора G.

(3.1) Количество команд склеивания, которые можно определить для аргументов и результатов G, не превышает $\min(m, |\cup A_k(G)|)$, где $k = 1,...,m$.

(3.2) Склеивание аргумента G с некоторой собственной локальной переменной может воспрепятствовать склеиванию аргумента с результатом,

если результат определяется до последнего использования локальной переменной.

3.1. Определение кандидатных пар в первичных операторах

Первичными являются предикат равенства, вызов предиката и оператор разложения. Рассмотрим на примерах правила построения кандидатных пар для этих операторов.

Кандидатной парой для предиката равенства a = b+c является <b,c: a>, что допускает возможность выполнения одной из команд склеивания <c: a> или <b: a>.

Пусть имеется определение предиката со следующим заголовком: Foo (**int** a,b: **int** a′, b′). Здесь, правилами языка определены следующие склеивания параметров Foo: <a: a′> и <b: b′>. На основе этой информации для оператора вызова Foo (p1+p2, p3+p4: r1, r2) строятся следующие кандидатные пары <p1, p2: r1> и <p3, p4: r2>.

Результатом разложения непустой последовательности (или множества) являются элемент и подпоследовательность (подмножество), которую можно склеить с исходной последовательностью (множеством).

Пример 3.

```
seq int a;
...
int c;
seq int b;
a→(|int c, seq int b)
```

В примере 3 можно склеить переменные a и b.

3.2. Склеивание в операторах ветвления.

Рассмотрим оператор ветвления G, имеющий форму простой функции, т.е. когда каждая ветвь оператора определяет одно и тоже множество результатов.

Пример 4.

```
if a < 5 then x = a else x = b end;
d = x*x;
```

В примере 4 при независимом склеивании аргументов и результатов на разных ветвях получим взаимоисключающие команды <a: x> и <b: x>. Во избежание подобных конфликтов построение кандидатных пар для резуль-

татов оператора ветвления G реализуется для него в целом на основе анализа информационных связей результатов и аргументов G. Во множество кандидатов, определенных по отношению к G для некоторой результирующей переменной x, войдут все аргументы G, от которых зависит x на ветвях G. Кандидатной парой для оператора ветвления G в примере 4 является <a,b: x>.

Склеивать собственные локалы ветвей между собой и с аргументами можно независимо на каждой ветви.

Рассмотрим построение кандидатных пар для оператора ветвления.

Пример 5.

```
if x < 5 then y = sqrt(x) || q = x*x  else  y = x*x ; q = y*x*b end;
z = q + y;
```

В примере 5 переменную y нельзя склеить с x по ветви **else**, поэтому x не войдет в кандидатную пару для y. В кандидатную пару для некоторой результирующей переменной оператора ветвления не будем включать аргументы, которые нельзя склеить с этой результирующей переменной, хотя бы на одной ветви.

Пример 6.

```
if a < 5 then c = b || e =b  else  c = a || e =b  end;
d = c*e;
```

Пример 6 показывает, что выбор различных вариантов склеивания аргументов и результатов может влиять на количество склеиваний. В примере возможны два варианта множеств кандидатных пар: {< b: c >} и {<a: c >, <b: e>}.

Рассмотрим следующий фрагмент:

Пример 7.

```
if a < 5 then
    c = b ||
    if a>2 then x=sqr(a) else x=a end          //(1)
else
    c = a || x =b
end;
d = c*x;
```

В примере 6 бессмысленно строить кандидатные пары для переменной x внутри вложенного в ветвь **then** оператора ветвления (1), так как на уровне оператора (1) нет полной информации о зависимости x от аргументов. По-

этому если результирующие переменные G являются также результирующими для некоторого объемлющего оператора ветвления, то для таких переменных нельзя строить кандидатные пары — такое построение будет проходить на уровне объемлющего оператора ветвления.

Для операторов ветвления, имеющих форму гиперфункции, склеивание переменных будем проводить на каждой ветви гиперфункции независимо, так как каждая ветвь определяет независимый набор результирующих переменных.

3.3. Склеивание переменных в параллельном операторе

Пример 8.

G(a: y)||H(a,b: x)

Переменная a используется в обеих альтернативах параллельного оператора. Чтобы применить команду склеивания <a: y>, необходимо перед параллельным оператором запомнить значение переменной a, поскольку оно используется для вычисления H(a,b: x). Однако при замене параллельного исполнения двух компонент параллельного оператора на последовательное: H(a,b: x); G(a: y) с перестановкой операторов, команду <a: y> можно применить без предварительного сохранения значения a. Рассмотрим преобразование, заменяющее параллельный оператор на блок с последовательным исполнением, в котором для проведения склеиваний мы жертвуем параллелизмом.

Рассмотрим параллельный оператор G вида G_1 ||...|| G_m. Для каждой компоненты G построим кандидатные пары $<A_k(G_i), y_k>$, где $A_k(G_i)$ — переменные из $A(G_i)$, используемые при вычислении k-го результата y_k в G_i (i=1..m). Входящие в A(G) аргументы назовем **уникальными,** если они используются только в одной альтернативе, а аргументы, использующиеся в более чем одной альтернативе, назовем **общими**.

Определим следующие критерии эффективности склеивания переменных. Если результирующие переменные одного типа, то будем добиваться максимально возможного количества пар кандидатов. Если результирующие переменные разного типа, причем, размер занимаемой ими памяти определен однозначно, то будем склеивать переменные так, чтобы размер занимаемой памяти склеенных переменных был максимален. В общем случае, будем использовать в качестве критерия систему приоритетов, когда в первую очередь склеиваются переменные динамического размера, затем последовательности и множества, и в последнюю — те переменные, для

которых размер занимаемой памяти определен однозначно, в порядке убывания их размеров.

Очевидно, что склеивание некоторого уникального аргумента из $A_k(G_i)$ с y_k не препятствует параллельному исполнению других компонент G, поэтому будем рассматривать компоненты, в которых имеются кандидатные пары с общими аргументами. Склеивание общего для нескольких компонент аргумента с одним из результатов не позволяет сохранить параллелизм между этими компонентами. Поэтому заменим параллельное исполнение последовательным, причем будем искать расстановку операторов с максимальной эффективностью склеивания.

Сгруппируем компоненты G, в кандидатных парах которых есть общие аргументы. При этом оператор G_i будет принадлежать некоторой группе Г, если существует хотя бы один оператор из Г, с которым G_i имеет общие переменные-кандидаты. Компоненты, принадлежащие разным группам, можно исполнять параллельно, так как по построению у них нет общих аргументов.

Найдем для каждой группы порядок исполнения компонент, допускающий склеивание переменных с максимальной эффективностью. В большинстве случаев каждая группа будет содержать небольшое количество компонент, поэтому расстановку можно проводить полным перебором всех возможных вариантов. Менее затратные способы склеивания переменных в параллельных операторах с общими аргументами, в том числе учитывающих условие (3.1), будут рассмотрены в дальнейшем.

Пример 9.

G(a,b: y_2) || H(a: y_1) || F(a,b,c: y_3) || E(a,b,c,d: y_4)

Для параллельного оператора из примера 9, лучшим порядком следования операторов будет:

E(a,b,c,d: y_4); F(a,b,c: y_3); G(a,b: y_2); H(a: y_1)

при котором можно провести 4 склеивания: <a: y_1>, <b: y_2>, <c: y_3>, <d: y_4>.

Рассмотрим ситуацию, когда в параллельном операторе определяются результаты некоторого объемлющего оператора ветвления.

Пример 10.

if a<b **then**
G(a,b: y_2) || H(a: y_1) || F(a,b,c: y_3) || E(a,b,c,d: y_4)
else
G(b: y_2) || H(a,b: y_1) ||F(a,b,c: y_3) || E(a,b,c,d: y_4)
end

В примере 10 лучшие варианты склеивания на разных ветвях конфликтуют между собой:

	Лучший порядок следования	Лучшее склеивание на ветви
Ветвь **then**	H(a: y_1); G(a,b: y_2); F(a,b,c: y_3); E(a,b,c,d: y_4)	<a: y_1>, <b: y_2>, <c: y_3>, <d: y_4>
Ветвь **else**	G(a: y_2);H(a,b: y_1); F(a,b,c: y_3); E(a,b,c,d: y_4)	<a: y_2>, <b: y_1>, <c: y_3>, <d: y_4>

В таких случаях необходимо отказаться от построения команд склеивания, конфликтующих со склеиванием на другой ветви.

3.4. Применение команд склеивания

Для первичных операторов G, где построены кандидатные пары <A_k(G), y_k>, определим команду <x: y_k> выбором любого x из A_k(G). Для оператора ветвления выберем такие команды, которые обеспечивают максимальное количество склеенных переменных при сохранении максимального количества подконструкций с параллельным исполнением.

Построение полученных команд реализуется в процессе рекурсивного обхода рассматриваемого определения предиката. В каждой конструкции будем обходить все подконструкции в глубину до достижения всех первичных операторов. В случае блока, в том числе полученного преобразованием параллельного оператора, обход начнем с последнего оператора. Для оператора ветвления установим следующий порядок обхода**:** сначала обходятся все ветви, затем заголовок оператора. В процессе обхода при достижения первичного оператора проведем замены в определении предиката в соответствии с командой склеивания.

Результатом применения к определению предиката произвольной команды склеивания, будет эквивалентная программа, так как по построению и в случае предложенного порядка обхода определения предиката переменные из A_k(G) не встречаются далее по исполнению, с другой стороны, y_k не может быть переопределено. Таким образом, до оператора G определение

предиката останется неизменным, а начиная с оператора G и далее по исполнению произойдет замена названия переменной, что не изменит процесса и результатов вычислений.

3.5. Замечания

Общий алгоритм склеивания является линейным по отношению к числу переменных в определении предиката. Применение полного перебора для определения порядка исполнения компонент параллельного оператора с числом компонент n определяет оценку сложности этой части алгоритма как O(n!) для худшего случая, когда число общих аргументов больше n.

Для указанных пользователем склеиваний входных параметров предикатов с результирующими будем проверять наличие информационной связи между склеиваемыми переменными. При отсутствии такой связи склеивание квалифицируется как некорректное.

ЗАКЛЮЧЕНИЕ

В данной работе описан линейный алгоритм склеивания переменных, учитывающий ограничения и особенности операторов ветвления и параллельных операторов предикатной программы.

В дальнейшем предполагается решить следующие задачи:

- склеивание переменных структурных типов разной длины, в том числе массивов;
- нахождение менее затратного способа реорганизации параллельных операторов;
- определение взаимосвязей между склеиванием переменных и последующими трансформациями предикатной программы.

СПИСОК ЛИТЕРАТУРЫ

1. **Шелехов В.И.** Введение в предикатное программирование — Новосибирск, 2002. — 82 с. — (Препр. / ИСИ СО РАН; №100).
2. **Шелехов В.И.** Язык предикатного программирования. — Новосибирск, 2002. — 40 с. — (Препр. / ИСИ СО РАН; №101).
3. **Ершов А.П.** Введение в теоретическое программирование — М.: Наука, 1977. — 288 с.

4. **Поттосин И.В.** Направленные преобразования линейного участка // Языки и системы программирования. — Новосибирск, 1981. — С.14–28.
5. **Bacon D., Graham S., Sharp O.** Compiler transformations for high-performance computing // ACM Computing surveys. — 1994. — Vol. 26, No. 4 — P. 345–420.