Российская академия наук
Сибирское отделение
Институт систем информатики
им. А. П. Ершова

И.В Каблуков, В.И. Шелехов

РЕАЛИЗАЦИЯ СКЛЕИВАНИЯ ПЕРЕМЕННЫХ В ПРЕДИКАТНОЙ ПРОГРАММЕ

Препринт
167

Новосибирск 2012

Трансформация склеивания переменных есть замена в предикатной программе всех вхождений одной переменной на другую. На базе потокового анализа программы определяется корректный набор склеиваний переменных, сокращающий число переменных и устраняющий избыточные копирования.

Siberian Division of the Russian Academy of Sciences
A. P. Ershov Institute of Informatics Systems

I.V. Kablukov, V.I. Shelekhov

VARIABLE GLUINGS IN A PREDICATE PROGRAM

Preprint
167

Novosibirsk 2012

The variable gluing transformation is a replacement of all occurrences of a variable to another variable in a predicate program. The sound set of variable gluings is founded on the basis of data flow analysis of the program.

ВВЕДЕНИЕ[1]

Язык предикатного программирования P [2] — это язык функционального программирования, в котором сочетаются функциональный и операторный (предикатный) стили записи алгоритмов. Он обладает существенно большей выразительностью по сравнению с чисто функциональными языками. По синтаксису, набору операторов и операция язык приближен к стилю языков семейства C. Эффективность предикатных программ достигается применением системы оптимизирующих трансформаций, переводящих программы на язык императивного расширения языка P.

Базовыми трансформациями являются склеивание переменных, реализующее замену нескольких переменных одной; замена хвостовой рекурсии циклом; подстановка определения предиката на место его вызова; кодирование рекурсивных структур с помощью массивов и указателей.

Склеивание переменных [1] — это замена (сохраняющая эквивалентность) в тексте программы всех вхождений одной переменной на другую. Значительный эффект достигается при склеивании структурных переменных, таких как массивы и списки, поскольку склеивание обычно позволяет избежать копирования структур.

В отличие от задачи экономии памяти [3], склеиванию в программе подлежат результаты с аргументами, аргументы с локалами и локалы с результатами, причем типы склеиваемых переменных должны совпадать. Набор склеиваний может быть частично задан пользователем. Необходимо проверить его корректность и дополнить. В языке P запрещено присваивание вида: x:=op(x, y). Вместо этого используется присваивание x:=op(x1, y) в предположении, что переменная x1 должна быть склеена с переменной x при трансляции на императивное расширение языка P. 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 оператор a:=a, удаляемый из программы. Типы склеиваемых переменных должны совпадать. В дополнение к этому переменная параметрического типа T(n+1) может быть склеена с переменной типа T(n) в случае возрастания n на 1.

[1] Работа выполнена при финансовой поддержке РФФИ, грант № 12-01-000686.

1. ОБЩАЯ СХЕМА РЕАЛИЗАЦИИ

Система предикатного программирования включает: front-end (реализующий синтаксический и семантический анализ), потоковый анализатор, генератор условий корректности при дедуктивной верификации, оптимизирующий трансформатор и др. компоненты.

Полная программа состоит из набора программ-определений предикатов. На первом этапе оптимизирующий трансформатор заменяет хвостовую рекурсию циклом во всех рекурсивных программах с хвостовой рекурсией. Далее реализуется подстановка тел программ на место вызовов для программ, разрешенных для подстановки. На третьем этапе для всех программ проводится склеивание переменных. Наконец, рекурсивные структуры данных кодируются с помощью массивов и указателей.

Потоковый анализатор строит слои программы, а также регионы склеивания и *пост-аргументы* для каждого оператора. Пост-аргументами оператора являются переменные, используемые при дальнейшем исполнении после завершения исполнения оператора. Склеивание переменных реализуется в оптимизирующем трансформаторе с помощью алгоритмов уточнения регионов.

Регион склеивания для оператора G есть набор аргументов и результатов одного типа <x: y>, где x — список аргументов и y — список результатов оператора.

Пример. Пусть имеется оператор F с аргументами a, b, c, d, e и результатами f, g, h. Пусть переменные a, b, d и g, h имеют натуральный тип, а переменные c, e и f — массивы. Тогда для оператора F регионы склеивания таковы: <a, b, d: g, h> и <c, e: f>. Регион склеивания вида <a**:** g> , где a и g — переменные, является *командой склеивания*, определяющей замену переменной a на g.

Полная программа делится на *слои*: слой первого уровня содержит программы, не вызывающие другие программы, слой второго уровня содержит программы, вызывающие только программы из слоя первого уровня, и т.д. Рекурсивное кольцо — набор программ, каждая из которых может быть вызвана по цепочке вызовов программ из этого кольца, начиная с вызова внутри себя. Каждое рекурсивное кольцо определяет отдельный слой. *Рекурсивный вызов* — это вызов программы из того же рекурсивного кольца, в котором находится вызывающая программа. Построение регионов упорядочено по уровням слоев, начиная с первого, и реализуется обходами дерева программы сверху вниз и снизу вверх.

Для вызова функции, у которой результат склеивается с одним из аргументов, оператор, включающий вызов, при анализе представляется в виде оператора суперпозиции. Например, x = 1+foo(b) заменяется на foo(b: t); x = 1+t. Здесь t склеивается с b, x склеивается с t, и в итоге x склеивается с b.

2. ПОСТРОЕНИЕ ГРАФА ВЫЗОВОВ

Полная программа — это набор определений предикатов. *Граф вызовов программы* — это ориентированный граф. Вершины графа — предикаты. Вызов предиката A в теле предиката B определяет дугу в графе: B→A. При наличии вызова предиката A(...) мы говорим, что предикат A используется в *позиции вызова предиката*.

Всякий предикат имеет тип. По набору предикатов программы определяется список предикатных типов (СПТ). Для каждого типа из СПТ строится множество *заместителей* данного типа — это набор предикатов этого типа, встречающихся в позициях, отличных от вызовов предикатов. Среди таких позиций — вхождение имени предиката в качестве аргумента некоторого вызова. Такие предикаты могут стать заместителями, т.е. значениями переменных и выражений предикатного типа.

Допустим, в теле предиката B имеется вызов предиката в виде e(...), где e — выражение предикатного типа, причем e — не имя предиката. Тогда в граф вызовов добавляются дуги вида $B \rightarrow A_i$, где A_i — заместитель для типа выражения e. Если же e — имя предиката, то в граф вызовов добавляется только дуга B→e.

На первом проходе по программе строятся заместители предикатных типов. На втором проходе строится собственно граф вызовов.

Рекурсивное кольцо — набор программ, каждые две из которых могут вызвать друг друга по цепочкам вызовов программ из этого кольца.

Алгоритм построения слоев графа и рекурсивных колец. Предварительно вершины предикатов, вызывающих самих себя, помечаются как рекурсивные кольца; при этом дуги от вершин на себя удаляются из графа. В слой первого уровня записываются все предикаты, вершины которых не имеют исходящих дуг; эти вершины (вместе с дугами на них) удаляются из графа. Слои второго и следующих уровней для измененного графа строятся аналогично; вершины этих слоев удаляются из графа. Если на некотором шаге не осталось вершин без исходящих дуг, но граф не пуст, то это означает, что в нем есть циклы. Граф обходится по произвольному пути, пока не пройдет по одной вершине дважды. Все вершины участка пути от пер-

вого вхождения вершины до второго стягиваются в графе в одну вершину — рекурсивное кольцо. Все рекурсивные кольца, встречающиеся на этом участке, становятся частями нового кольца. Все входящие и исходящие дуги вершин, помещаемых в кольцо, приписываются новой вершине; внутренние дуги между вершинами кольца удаляются. Если у новой вершины (рекурсивного кольца) нет исходящих дуг, то все предикаты рекурсивного кольца записываются в новый рекурсивный слой, а сама вершина удаляется из графа. Далее алгоритм возвращается к построению обычных слоев. Если у новой вершины (рекурсивного кольца) есть исходящие дуги, то продолжается движение по избранному пути до очередного самопересечения.

Алгоритм завершается при пустом графе. Сложность алгоритма в худшем случае $O(n^3)$, где n — число предикатов.

3. ПОСТРОЕНИЕ ПОСТ-АРГУМЕНТОВ И РЕЗУЛЬТАТОВ

Для произвольного оператора D определим рекурсивную программу $DF(D, ArgP_D\mathbf{:}\ Arg_D, R_D)$, вычисляющую аргументы Arg_D и результаты R_D оператора D, а также аргументы, результаты и пост-аргументы для всех вложенных операторов; $ArgP_D$ обозначает ранее вычисленные пост-аргументы оператора D.

Для оператора суперпозиции A = B; C программа $DF(A, ArgP_A\mathbf{:}\ Arg_A, R_A)$ представляется следующей последовательностью операторов:

$ArgP_C = ArgP_A$;
$DF(C, ArgP_C\mathbf{:}\ Arg_C)$
$ArgP_B = ArgP_A \cup Arg_C$;
$DF(B, ArgP_B\mathbf{:}\ Arg_B)$
$Arg_A = Arg_B \cup Arg_C$;
$R_A = \{R_B \setminus Loc_B\} \cup R_C$;

Для условного оператора A = **if** (E) B **else** C и параллельного оператора A = B || C программа $DF(A, ArgP_A\mathbf{:}\ Arg_A, R_A)$ представляется следующей последовательностью операторов:

$ArgP_B = ArgP_A$;
$ArgP_C = ArgP_A$;
$DF(B, ArgP_B\mathbf{:}\ Arg_B)$
$DF(C, ArgP_C\mathbf{:}\ Arg_C)$
$Arg_A = Arg_B \cup Arg_C$;
$R_A = R_B \cup R_C$;

4. ПОСТРОЕНИЕ РЕГИОНОВ СКЛЕИВАНИЯ

Для построения регионов склеивания программа обходится снизу вверх. В дереве предикатной программы нижними операторами являются операторы присваивания и вызовов предикатов. Для этих операторов строятся регионы склеивания определенным ниже образом. После обходятся объемлющие операторы суперпозиции, параллельные и условные операторы, регионы которых строятся на основе уже определенных регионов подоператоров.

Для оператора **присваивания** регион строится с набором его аргументов в левой части и переменной в правой части.

Предполагается, что для предиката, вызываемого в операторе **вызова** предиката, уже построены итоговые регионы склеивания результатов с аргументами. Подставляя соответствующие аргументы и результаты вызова в эти регионы, получаются регионы данного оператора вызова.

В рекурсивном предикате для хвостового рекурсивного вызова регионов не строится, т. к. при замене хвостовой рекурсии вызов заменяется мультиприсваиванием аргументам предиката новых значений, и поэтому соответствующий набор регионов пуст.

В случае рекурсивного кольца с числом предикатов больше одного регионы для рекурсивных вызовов строятся с использованием лишь априорных склеиваний вызываемых предикатов. Фактически это означает требование к пользователю явным образом задавать все склеивания результатов с аргументами для таких предикатов.

Пример

Foo (**int** a, b**: int** a’, b’)
{ ... }

В соответствии с правилами языка Р использование имен a’ и b’ означает, что переменная a’ должна быть склеена с a, а переменная b’ — с b. Подстановка тела предиката Foo на место вызова Foo (p1+p2, p3+p4**:** r1, r2) с раскрытием мультиприсваиваний дает последовательность: a = p1+p2; b = p3+p4; {…} r1 = a; r2 = b. Переменные a и b становятся локалами. Строятся следующие регионы: <p1, p2**:** a>, <p3, p4**:** b>, <a**:** r1> и <b**:** r2>.

В подоператорах **параллельного оператора** все результаты различны, а аргументы делятся на уникальные — участвующие только в одном подоператоре, и общие — участвующие в более чем одном подоператоре. Уникальные аргументы можно склеить с результатами, а общие аргументы

склеивать нельзя, т.к. они используются в других подоператорах. Однако при замене параллельного выполнения на последовательное можно склеить также и общие аргументы с результатами.

Регионы для параллельного оператора строятся по регионам его подоператоров. Алгоритм построения следующий. Если регион содержит и уникальные и общие аргументы, то все общие удаляются. Далее для регионов только с общими аргументами выбирается аргумент, встречающийся в наименьшем количестве регионов. Выбираются регионы, содержащие его, и удаляются все, кроме одного. В оставшемся регионе из левой части удаляются все аргументы, кроме данного общего. Таким способом обрабатываются все регионы подоператоров; получившиеся регионы становятся регионами параллельного оператора. Отметим, что, следуя такому алгоритму, мы можем потерять возможные варианты склеивания.

В **условном операторе,** если есть регионы с разными результатами и одинаковым аргументом, то этот аргумент удаляется из этих регионов. Данное правило не относится к регионам с локалами ветвей в правых частях. Например, у одной ветви есть регион <a: x>, у другой — <a: d>, где a — аргумент условного оператора, x — результат, а d — локал второй ветви. Оба этих склеивания можно провести, т. к. локал, в отличие от результата ветвей, не будет использоваться после условного оператора, поэтому две переменные с одинаковым именем не будут использоваться одновременно. Оставшиеся регионы с одинаковым результатами объединяются, образуя один регион с данным результатом в правой части и объединением аргументов в левой.

Для **оператора суперпозиции** регионы строятся как объединение регионов подоператоров.

5. ПОСТРОЕНИЕ КОМАНД СКЛЕИВАНИЯ

Команды склеивания строятся в оптимизирующем трансформаторе уточнением регионов склеивания, построенных потоковым анализатором. Сначала регионы уточняются набором *априорных склеиваний* и *запретов*, заданных в исходной программе. Пользователь может задать в заголовке предиката склеивание результата с аргументом. В этом случае имя результата есть имя аргумента с добавлением штриха в конце. Команда склеивания (или запрета склеивания) может быть также задана прагмой. Проверя-

ется, что априорные склеивания содержатся в регионах, т. е. для каждого априорного склеивания есть такой регион склеивания, в правой части которого содержится результат априорного склеивания, а одна из переменных левой части — аргумент априорного склеивания. В противном случае выдаются сообщения о некорректном задании априорных склеиваний. Запрет на склеивание — пара переменных, означающая, что вторую переменную нельзя склеить с первой. Проверяется, что данная пара переменных не принадлежит ни одному региону. Если есть такой регион склеивания, в правой части которого содержится вторая переменная запрета склеивания, а одна из переменных левой части региона — первая переменная запрета склеивания, то эта первая переменная удаляется из региона.

Все получившиеся регионы с одним результатом и аргументом считаются командами склеивания. Для регионов с несколькими аргументами и одним или несколькими результатами произвольным образом выбираются команды склеивания с одним результатом и одним аргументом.

Далее обрабатываются команды, содержащие локалы предиката. Если команда имеет вид <аргумент: локал>, то во все команды, кроме текущей, аргумент подставляется на место локала.

После построения полного набора команд склеивания программа обходится сверху вниз, склеивая переменные исходя из регионов склеивания. Замена параллельных операторов на последовательные дает дополнительные возможности для склеивания.

Алгоритм. Для параллельного оператора из построенных команд склеивания выбираются те, что содержат его результаты в правых частях. Рассматривается каждая выбранная команда склеивания. Если слева — уникальный аргумент параллельного оператора, то склеивание можно провести без конфликтов с другими подоператорами. Если слева — общий аргумент параллельного оператора, а справа — результат параллельного оператора, то необходимо трансформировать параллельный оператор. Выбираются все подоператоры с этим аргументом, кроме того, в результатах которого содержится результат из этой команды, и исходный параллельный оператор заменяется суперпозицией **A;B** , где **A** — параллельный оператор из выбранных подоператоров, а **B** — исходный параллельный оператор без выбранных подоператоров. В операторе **B** этот аргумент станет уникальным, и с ним можно будет склеить результат.

6. ПРИМЕР

Работу алгоритма склеивания переменных покажем на примере предикатной программы нахождения целочисленного квадратного корня:

```
sq1(nat x, k, n : nat m)
{     nat p = n + 2* k + 1;
      if (x < p) m = k else  sq1(x, k + 1, p: m)
}
```

Построение регионов реализуется, начиная с простых операторов. Для операторов присваивания строятся регионы <n: p> и <k: m>. В первом операторе k — пост-аргумент, поэтому с ним склеивать нельзя, и он не может находиться в регионе. Для рекурсивного вызова регионы не строятся. Далее на условном операторе объединяются регионы его ветвей. В результате единственным регионом условного оператора станет <k: m>. Тело программы — оператор суперпозиции оператора присваивания с условным оператором. Регионы оператора суперпозиции строятся объединением регионов его подоператоров. В результате для тела программы и для программы sq1 в целом получим регионы <n: p> и <k: m>. Эти регионы являются итоговыми командами склеивания. Отметим, что предварительно надо было бы заменить вхождения локала p в других командах, однако таковые отсутствуют.

Процесс склеивания реализуется обходом программы. Вхождения в программе переменных из правых частей команд склеивания заменяются соответствующими переменными из левых частей, т.е. переменная p заменяется на n, а переменная m — на k.

Итоговая программа, в которой произведены склеивания <n: p> и <k: m>:

```
sq1(nat x, k, n)
{     n = n + 2* k + 1;
      if (x < n) k = k else  sq1(x, k + 1, n: k)
}
```

В дальнейшем оператор k = k удаляется из программы.

СПИСОК ЛИТЕРАТУРЫ

1. Петров Э.Ю. Склеивание переменных в предикатной программе // Методы предикатного программирования. Новосибирск, 2003. С. 48-61.
2. Шелехов В.И. Введение в предикатное программирование. Новосибирск,

2002. 82с. (Препр. / ИСИ СО РАН; N 100).
3. Ершов А.П. Введение в теоретическое программирование. М.: Наука, 1977. 288с.

И.В Каблуков, В.И. Шелехов

РЕАЛИЗАЦИЯ СКЛЕИВАНИЯ ПЕРЕМЕННЫХ В ПРЕДИКАТНОЙ ПРОГРАММЕ

Препринт
167

Рукопись поступила в редакцию 05.10.2012
Редактор Т. М. Бульонкова
Рецензент В.В. Михеев

Подписано в печать 08.11.2012
Формат бумаги 60 × 84 1/16 Объем 0.8 уч.-изд.л., 0.88 п.л.
Тираж 50 экз.

Центр оперативной печати «Оригинал 2»
г.Бердск, ул. О. Кошевого, 6, оф. 2, тел. (383-41) 2-12-42