Р. И. Идрисов

ПРОТЯГИВАНИЕ КОНСТАНТ В ГРАФЕ IR2 ВНУТРЕННЕГО ПРЕДСТАВЛЕНИЯ ЯЗЫКА SISAL

ВВЕДЕНИЕ

Потоковый язык SISAL 3.2 [1], разрабатываемый в Институте систем информатики им. А. П. Ершова, является функциональным языком однократного присваивания. Основная часть оптимизаций в компиляторе языка SISAL выполняется на уровне представления IR2, которое является иерархическим потоковым бесконтурным графом программы. Представление состоит из вершин и портов, где вершины обозначают операции, а порты используются для обозначения начала и окончания дуги передачи значения между операциями. В данной статье пойдёт речь о протягивании значений, алгоритме оптимизации, направленном на подстановку значений, которые могут быть вычислены до исполнения программы. В контексте компилятора Sisal алгоритмы протягивания констант используются не только как оптимизирующие преобразования, а ещё и как алгоритмы анализа, которые направлены на определение возможных значений, известных на стадии компиляции программы. Эта информация используется другими оптимизирующими преобразованиями компилятора. Речь пойдёт не только о случае, когда значение может быть однозначно определено, но и когда есть некоторый набор или диапазон возможных значений. Названия глав данной статьи соответствуют характеру распространяемой информации.

1. ПРОТЯГИВАНИЕ ЗНАЧЕНИЙ

Этот алгоритм является самым простым из рассматриваемых. Он направлен на вычисление константных значений, которые могут быть получены до компиляции.

Алгоритм перебирает порты графа IR2 в порядке вычисления операндов и каждому из портов сопоставляет вычисленное значение переменной, если таковое может быть найдено. Значение может быть найдено, если:

- порт является входным, а соответствующий выходной порт принимает константное значение;

- порт является выходным, вершина соответствует простой операции, и входные порты этой вершины также определены. Под простой операцией подразумевается функция, которая, действуя на константные операнды, возвращает значение целого или булевского типа, например: сложить, вычесть, умножить.

Этот алгоритм может не найти некоторых константных значений, реально возникающих при исполнении программы.

Утверждение: подстановка значения, обнаруженного таким образом, является эквивалентным преобразованием программы.

Доказательство: если подстановка значения преобразует программу в неэквивалентную, значит, существует набор данных, для которого результат исполнения программы отличается от результата исполнения исходной программы. Пусть A_i – такой набор входных данных, B_k и B_k’ – результаты программ до и после подстановки, а K_j – набор подставленных значений. Выделим подставленные значения как входные параметры программы. В таком случае мы получаем две программы с эквивалентными графами вычислений. Эти программы на наборе входных данных (A_i, K_j) должны приводить к одинаковому результату, что противоречит нашему предположению о существовании набора данных A_i, приводящего к различным результатам вычислений.

Особые случаи возникают при рассмотрении сложных вершин, имеющих неявную внутреннюю структуру (select, for). В этих случаях информация распространяется по неявным связям внутри составной вершины и объединяется в случае нескольких возможных путей исполнения, но эта особенность не влияет на доказательство рассмотренного утверждения.

Рассмотрим пример:

```
function ts_411(returns integer)
  let
 i:=1
  in
   if i>0 then 1
   else 0
   end if
  end let
end function
```

В этом случае граф IR2 будет выглядеть так:

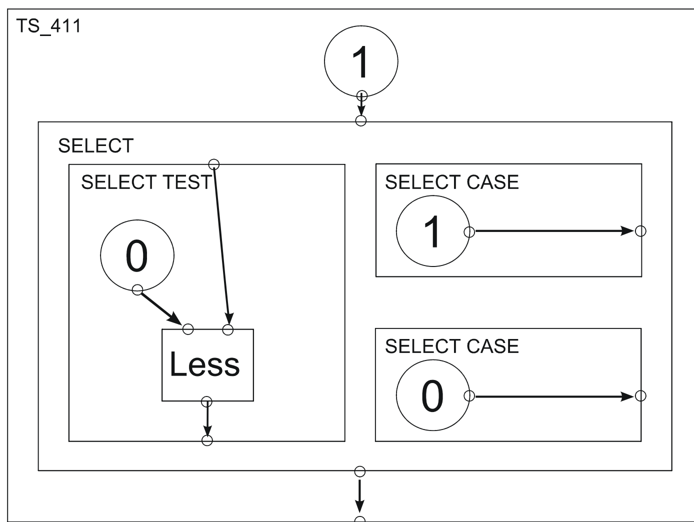

Рис. 1. Граф IR2 для функции ts_411

В данном случае. Информация о константном значении условия может показать компилятору, что в коде функции много лишнего «мёртвого» кода, рассмотрим на данном примере распространение информации о константных значениях. В процессе анализа такая информация определяется для каждой дуги подграфа. После того, как определено значение выходного порта подграфа Select Test, можно утверждать, что всегда будет выполняться только первая ветвь Select Case, а функция будет иметь константное выходное значение (рис. 1).

Информация о том, что функция принимает константное значение, должна быть выражена в терминах графа IR2. Для того, чтобы она могла быть использована любыми другими алгоритмами анализа или оптимизации. Для этого в граф добавляются вершины типа literal (константа) там, где значение постоянное (рис. 2). Исключение составляют те рёбра, которые идут из вершин типа literal, поскольку такое преобразование для них бессмысленно. Граф после подстановки констант можно увидеть на рис. 3. Константа булевского типа со значением «истина» (true) обозначена буквой «T».

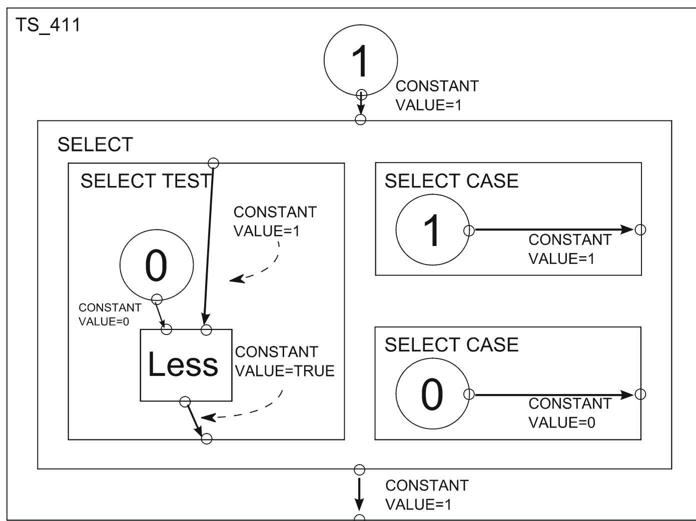

Рис. 2. Граф функции, дополненный алгоритмом анализа

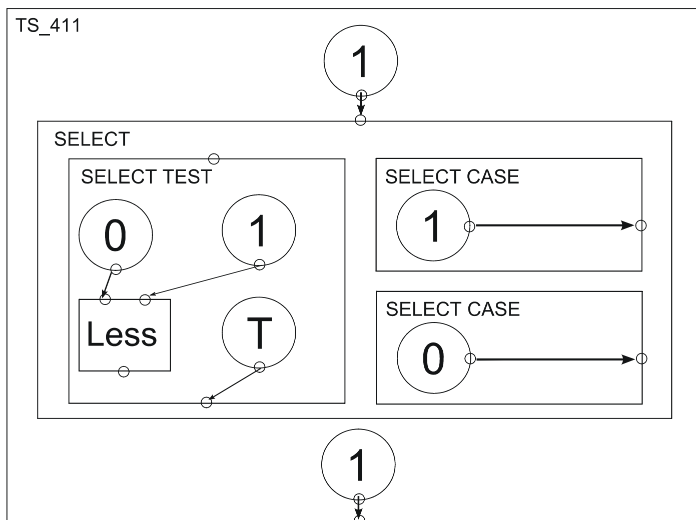

Рис. 3. Граф функции после подстановки константных значений

Этот алгоритм анализа имеет сложность, линейно зависящую от числа вершин графа функции O(N), поскольку для каждой вершины выполняется конечное количество операций, которое не зависит от общего числа вершин и других параметров программы, а является лишь особенностью реализации алгоритма.

2. ПРОТЯГИВАНИЕ МУЛЬТИЗНАЧЕНИЙ

Этот алгоритм отличается от предыдущего представлением данных. Здесь распространяется информация о множестве возможных значений, которое может быть задано при помощи перечисления. Имеется в виду, что здесь обрабатываются перечислимые конечные множества целых чисел.

Рассмотрим пример:

```
function ts_412(i : integer returns integer)
   let j := if i>0 then 1
     else 0
     end if
   in
     if j<2 then 1
     else 0
     end if
   end let
end function
```

В этом случае предыдущий алгоритм не сработает, поскольку нельзя сказать, что значение j всегда одно, но код очевидно «мёртвый», поскольку второе условие всегда истинно.

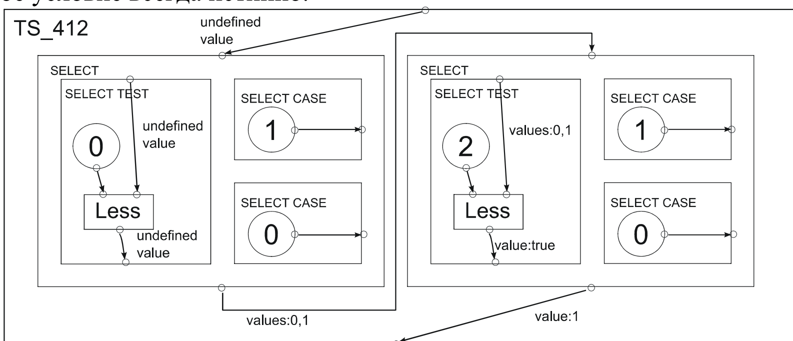

Рис. 4. Протягивание мультизначений

Подстановка производится только в том случае, когда полученное значение оказывается единственным, а записи о множестве возможных значений могут быть использованы другими оптимизирующими алгоритмами компилятора.

Количество возможных значений в данном случае ограничено количеством вершин типа literal (константа) с различным значением в графе программы, это число может стремиться к общему числу вершин графа программы. Сложность вычисления простой операции для множественных значений пропорционально их количеству. Таким образом, сложность алгоритма ограничена $O(N^2)$, где N – общее число вершин графового представления программы. Это оценка сверху; в реальных же случаях общее количество ветвлений программы ограничено константой и практически не зависит от общего числа вершин; если это предположение верно, сложность алгоритма – O(N).

3. ПРОТЯГИВАНИЕ ДИАПАЗОНОВ

Этот алгоритм отличается от предыдущего также представлением данных. Здесь, кроме константных значений, могут фигурировать интервалы значений, но не множество значений.

Информация о возможных значениях переменной представляет собой интервал [a;b] либо может быть не определена в случае, если область значений не может быть вычислена в процессе компиляции, либо не может быть записана в виде одиночного интервала такого типа.

Алгоритм анализа практически остаётся без изменений за исключением того, что вершины, порождающие множества значений (например, Scatter) теперь могут служить источником информации для алгоритма анализа. Конечно, такую возможность можно было реализовать и в рамках предыдущего алгоритма, но это не имело бы особого смысла, поскольку по точности он бы был таким же, как алгоритм, описываемый в следующем разделе (протягивание мультидиапазонов), но по скорости работы и объёму требуемых ресурсов явно уступал бы ему.

В данном случае требуется доопределение простых операций на операнды, представленные не одиночными значениями, а интервалами. Например, вычитание двух значений с возможными диапазонами [10..20] и [3..7] даст значение с возможным диапазоном [3..17].

Рассмотрим пример программы:

```
function ts_413(returns array[integer])
   for i in 1, 100 repeat
   s:=if i>0 then 1
     else 0
     end if
   returns array of s
   end for
end function
```

В этом случае предыдущие алгоритмы не нашли бы избыточный код, который здесь, очевидно, присутствует.

Сложность этого алгоритма также O(N), поскольку количество операций, выполняемое для каждой из вершин графа, ограничено константой, не зависящей от общего числа вершин (N).

4. ПРОТЯГИВАНИЕ МУЛЬТИДИАПАЗОНОВ

Протягивание мультидиапазонов даёт уточнение анализа в несколько экзотических случаях. Например, это происходит, когда циклическая переменная изменяется по набору интервалов или имеется ветвление в программе, которое различным образом определяет массив, используемый далее в программе. Сам факт уточнения данных в процессе анализа без реальной возможности это использовать не представляет пользы. Например, может встретиться условие, которое выполняется только при значениях, находящихся между определёнными диапазонами (предыдущий алгоритм из-за объединения интервалов в один не мог сохранить эту информацию).

Рассмотрим следующую программу:

```
function ts_414(returns array[integer])
  s1:=for I in 1, 10 returns stream of I;
  s2:=for I in 30,40 returns stream of I;
  s3:=s1 || s2;
  for I in s3 repeat
   s:=if I=20 then 1
         else 0
         end if
  returns sum of s
  end for
end function
```

В этом примере код не менее очевидно является избыточным, но установить это при помощи предыдущих алгоритмов анализа не представляется возможным.

Сложность этого алгоритма, как и в случае мультизначений, ограничена $O(N^2)$, здесь источниками диапазонов могут служить вершины типа scatter, количество которых ограничено сверху общим числом вершин графа программы N.

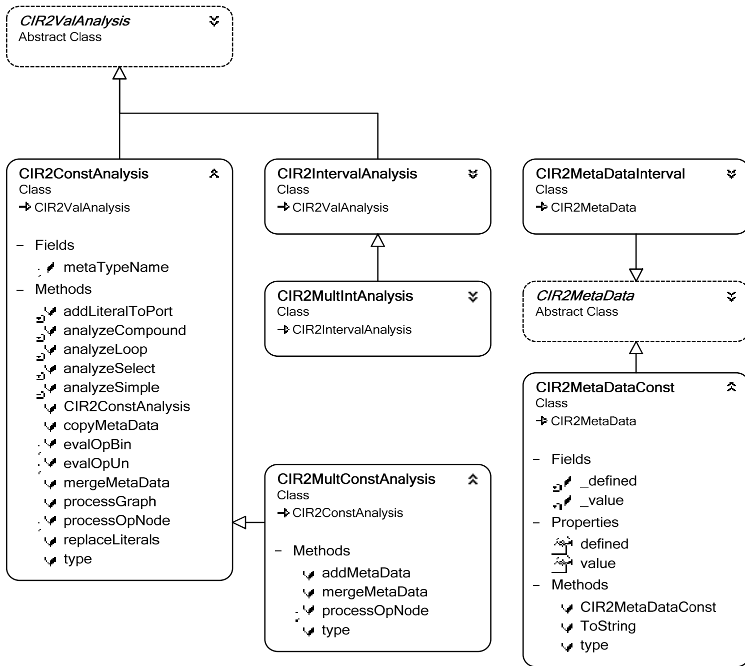

Рис. 5. Структура объектов, осуществляющих анализ

Такое большое количество алгоритмов, принципиально отличающихся только лишь представлением данных, нужно для проведения различных тестов и исследований, на которые и направлена система SFP, частью которой является компилятор Sisal.

Реализация четырёх рассмотренных алгоритмов анализа выполнена таким образом, что для добавления каждого следующего класса требовалось изменить только небольшие методы.

На рис. 5 изображены классы преобразований и классы метаданных, которые при этом используются. Классы для анализа значений и мультизначений используют один и тот же тип метаданных, только с тем отличием, что в случае одиночных значений такая запись может содержаться только одна для любого порта IR2. Для классов множественных значений и диапазонов потребовалось изменить алгоритм добавления метаданных порта IR2, алгоритм слияния значений метаданных и алгоритм обработки простой операции (преобразующей известные значения в известные).

5. ПРОТЯГИВАНИЕ ДРУГОЙ ИНФОРМАЦИИ О ЗНАЧЕНИЯХ

Когда одно из значений, составляющих выражение, является неопределённым, рассмотренные алгоритмы не будут срабатывать, но в случае сокращения неизвестного значения, общее выражение может быть вычислено до исполнения программы.

Рассмотрим следующий пример:

```
function ts_415(n: integer returns integer)
  let
 i:=n+1
 k:=n+5
  in
   if i>k then 1
   else 0
   end if
  end let
end function
```

В этом примере избыточный код можно обнаружить при помощи прямой подстановки (forward substitution). Несложно придумать пример, где такой способ не сработает:

```
function ts_4151(n: integer returns integer)
let
  i,k:=if n>0 then
     n+1,
     n+5
       else
     n+2,
     n+4
```

```
          end if
    in
      if i>k then 1
      else 0
      end if
    end let
    end function
```

В данном случае прямая подстановка в чистом виде ничего не даст, если только не будет скомбинирована с какими-нибудь более сложными алгоритмами.

Для этого примера можно ограничиться алгоритмом протягивания значений, только в качестве значения использовать тройку *связанная переменная, операция связи, константа*. Это очень похоже на способ описания областей массивов при помощи триплетов, с той разницей, что здесь в качестве связанных переменных выступают не итераторы циклов, а просто некоторые переменные программы.

В этом случае информация распространится следующим образом:

Рис. 6. Протягивание информации о связанной переменной

Оценив действие алгоритма, несложно модифицировать пример так, чтобы программа сохранила свой смысл, но алгоритм не срабатывал:

```
function ts_4151(n: integer returns integer)
let
  i,k:=if n>0 then
     n+1,
     n+5
       else
     1,
     5
       end if
in
  if i>k then 1
  else 0
  end if
end let
end function
```

В этом случае уже не получится обойтись этим же алгоритмом с модифицированным представлением данных, потребуется анализ с учётом пути исполнения. Отметим также, что этим примером неидеальности алгоритма не заканчиваются; значение может зависеть от двух или трёх переменных, и в процессе его вычисления могут встречаться произвольные функции.

Сформулируем общий критерий: подстановку константы можно совершить на стадии компиляции тогда и только тогда, когда может быть однозначно вычислен статический срез по переменной подстановки. Здесь срез – это подмножество операторов и выражений исходной программы, которые прямо или косвенно влияют на значения, вычисляемые в точке критерия среза (в нашем случае это значения, для которых требуется определить возможность их подстановки на стадии компиляции), но не обязательно составляют исполняемую программу [3].

Становится ясно, что для более точного поиска инвариантных значений можно воспользоваться алгоритмами вычисления минимального статического среза.

Для построения однопроцедурных статических срезов программ задача сводится к нахождению всех достижимых вершин из точки критерия среза в графе программных зависимостей, что является верным и для графа IR2, поскольку операторы, вычисляемые в процессе получения значения, соответствуют операторным вершинам, достижимым при прохождении графа IR2 в направлении, обратном выполнению вычислений.

Построение статического среза программы, представленной в виде графа программных зависимостей, является задачей линейной сложности. Срез должен быть вычислен для каждого значения в программе, после чего по-

требуется дать заключение о возможности его вычисления до исполнения программы. Таким образом, сложность алгоритма O(V*(N+D)), где V количество значений, для которых требуется произвести вычисление среза, а N и D количество вершин и рёбер графа IR2. В случае, когда срез не является однозначно вычислимым, представление возможных значений в виде среза не является удобным для других алгоритмов оптимизации и вряд ли может быть использовано повторно.

ЗАКЛЮЧЕНИЕ

Алгоритмы, рассмотренные в параграфах 1-4, реализованы в рамках системы функционального программирования SFP ИСИ СО РАН. Сложность и эффективность алгоритмов для протягивания множественных значений и интервалов требуется проверить на реальных программах.

Задача построения среза является более общей по отношению к анализу значений в графе программных зависимостей программы, но такой вид анализа предоставляет результаты, которые сложно использовать в других алгоритмах оптимизации. Эффективность и сложность этого метода также требуется проверить на реальном классе задач.

СПИСОК ЛИТЕРАТУРЫ:

1. Касьянов В. Н., Бирюкова Ю. В., Евстигнеев В. А. Функциональный язык Sisal // Поддержка супервычислений и интернет-ориентированные технологии. – Новосибирск: ИСИ СО РАН, 2001. — С. 54–67.
2. Евстигнеев В. А., Серебряков В. А. Методы межпроцедурного анализа (обзор) // Программирование. – 1992. – № 3. – С. 4–15.
3. Касьянов В. Н., Мирзуитова И. Л. SLICING: Срезы программ и их использование. – Новосибирск, 2002. – 116 с.
4. Keryell R. et al. PIPS – A Workbench for Interprocedural Program Analyses and Parallelization / R. Keryell, C. Ancourt, F. Coelho, B. Creusillet, F. Irigoin, P. Jouvelot – Paris, 1996. – 24 p. – (Tech. Rep. / Centre de Recherche en Informatique. Ecole Nationale Sup'erieure des Mines de Paris)