

13. Анализ потоков данных

- Определение анализа потоков данных
- Достижимые определения и живые переменные
- Формализация задач анализа потоков данных
- Итеративный алгоритм для решения задач анализа потоков данных

Лекция 13. Анализ потоков данных

В этой лекции рассматриваются следующие вопросы:

- Определение анализа потоков данных
- Достижимые определения и живые переменные
- Формализация задач анализа потоков данных
- Итеративный алгоритм для решения задач анализа потоков данных

Анализ потоков данных

Задача:

- определение глобальных свойств программы
- проверка контекстных условий оптимизирующих преобразований

Идея:

- определение свойств исполнения программы для каждого пути в графе управления
- выделение общей части

Способ:

- итеративный алгоритм анализа потоков данных

Анализ потоков данных

Под анализом потоков данных понимают совокупность задач, нацеленных на выяснение некоторых глобальных свойств программы, то есть извлечение информации о поведении тех или иных конструкций в некотором контексте. Такая постановка задачи возможна по той причине, что язык программирования и вычислительная среда определяют некоторую общую, "безопасную" семантику конструкций, которая годится "на все случаи жизни". Учет же контекстных условий позволяет делать более конкретные, частные заключения о поведении той или иной конструкции; при этом такие заключения, вообще говоря, перестают быть верными в другом контексте. Например, общая семантика присваивания заключается в вычислении выражения, стоящего в правой части, и присваивании полученного значения в переменную, стоящую в левой части. Однако в случае, когда выражение в правой части не имеет побочных эффектов, а переменная в левой части более нигде не используется, данный оператор становится эквивалентен пустому.

Для того чтобы описать понятие контекста, снова обратимся к графу потока управления (см. лекции 11, 12). Понятно, что на смысл каждой конструкции может оказывать влияние любая конструкция, из которой в этом графе достижима данная. Отсюда следует, что для правильного учета контекста необходимо учесть влияние всех путей до данной вершины, сначала определив влияние каждого пути, а затем выделив общую часть. Задача осложняется тем, что при наличии контуров множество всех путей в графе управления становится бесконечным.

Далее в этой лекции будет рассмотрен общепринятый итеративный подход, который позволяет получить приближенное решение задач анализа потоков данных, а при определенных условиях это решение становится точным.

Пример

```
struct S {int a; int b};

int F (int n, struct S * v)
{
    int i, s = 0;

    for (i=0; i<n; i++)
    {
        int q = (v+i)->a - (v+i)->b;

        if (q < 0) s += (v+i)->a + (v+i)->b;
        else (v+i)->b = q;

        (v+i)->a = (v+i)->b;
    }

    return s;
}
```

Пример

Для демонстрации сути задач анализа потоков данных рассмотрим несколько примеров.

На иллюстрации приведен фрагмент программы. Вхождения одного и того же выражения $(v+i) \rightarrow b$, обведенные сплошной линией, являются эквивалентными. В то же время вхождение того же самого выражения, обозначенное пунктирной линией, не эквивалентно первым двум, поскольку else-часть условного оператора содержит разрушающее присваивание.

Понятно, что для выяснения эквивалентности данных выражений необходимо перебрать все пути и убедиться, что ни в одном из них значения переменных, входящих в выражения, не меняются.

Достижимые определения

```
struct S {int a; int b};

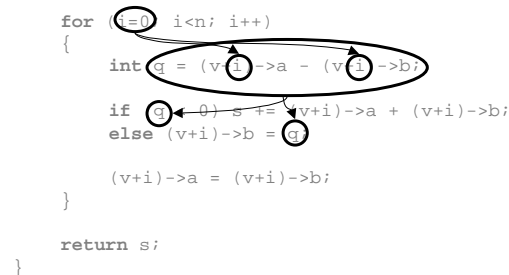
int F (int n, struct S * v)
{
    int i, s = 0;

    for (i=0; i<n; i++)
    {
        int q = (v->i->a - (v->i->b);

        if (q < 0) s += (v+i)->a + (v+i)->b;
        else (v+i)->b = q;

        (v+i)->a = (v+i)->b;
    }

    return s;
}
```



Достижимые определения

Достижимые определения являются одной из классических задач анализа потоков данных. Эту задачу можно сформулировать следующим образом:

для каждого вхождения переменной требуется определить множество присваиваний, такое, что для каждого из них существует путь, в котором между ним и данным вхождением отсутствуют другие присваивания той же переменной.

Неформально говоря, задача достижимых определений заключается в выяснении, где именно устанавливаются значения того или иного вхождения данной переменной.

На слайде показан пример программы, в котором выделены вхождения некоторых переменных и некоторые присваивания. Стрелки ведут от определений (присваиваний) к вхождениям переменных.

Видно, что решения этой задачи достаточно для построения представления программы с использованием def-use chains, которое необходимо для проведения многих оптимизаций (см. лекцию 11).

Живые переменные

```
struct S {int a; int b};

int F (int n, struct S * v)
{
    int i, s = 0;                                {n, v}

    for (i=0; i<n; i++)                            {i, s, n, v}
    {
        int q = (v+i)->a - (v+i)->b;                {i, s, n, v}

        if (q < 0) s += (v+i)->a + (v+i)->b;        {i, s, n, v, q}
        else (v+i)->b = q;

        (v+i)->a = (v+i)->b;                        {i, s, n, v, q}
    }

    return s;
}
```

Живые переменные

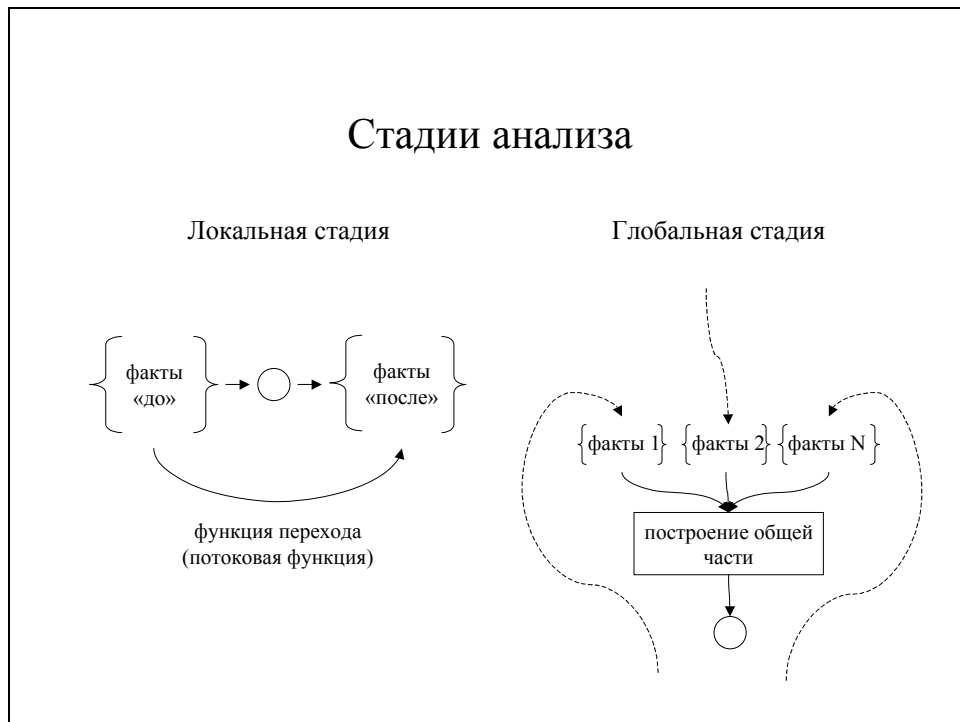
Живые переменные также являются классической задачей анализа потоков данных. В ней требуется для каждой вершины графа потока управления построить множество переменных, обладающих следующим свойством:

существует путь через данную вершину, начинающийся присваиванием данной переменной и кончающийся ее использованием, не содержащий иных присваиваний той же переменной.

Пример решения данной задачи для конкретной программы показан на слайде.

В общем случае, решение данной задачи играет важную роль в распределении регистров.

Стадии анализа



Стадии анализа

Из вышесказанного видно, что логически процесс решения задачи анализа потоков данных состоит из двух стадий, выполняемых одновременно.

Локальная стадия заключается в учете влияния отдельного оператора (группы операторов в вершине графа управления) в предположении, что уже имеется решение задачи анализа потоков данных *перед* этим оператором.

На глобальной стадии происходит решение задачи анализа для каждого пути, ведущего в данную вершину и затем выделение общей части всех таких решений.

Разметки и потоковые функции

$(X, <)$ - множество «фактов» с отношением частичного порядка

$G = (V, E, start, stop)$ - граф потока управления

$\mu : V \rightarrow X$ - разметка

Отношения на разметках:

$$\mu_1 = \mu_2 \Leftrightarrow \forall v \in V \mu_1(v) = \mu_2(v)$$

$$\mu_1 < \mu_2 \Leftrightarrow \forall v \in V \mu_1(v) < \mu_2(v)$$

$$\mu_1 \leq \mu_2 \Leftrightarrow \forall v \in V \mu_1(v) < \mu_2(v) \text{ или } \mu_1(v) = \mu_2(v)$$

$F : (V \rightarrow X) \rightarrow (V \rightarrow X)$ - функция перехода

μ_s - неподвижная точка $F \Leftrightarrow F(\mu_s) = \mu_s$

Разметки и потоковые функции

Опишем теперь общий подход к решению задачи анализа потоков данных более формально.

Зафиксируем некоторое частично упорядоченное множество "фактов" (утверждений о свойствах программы) X . Отображение μ , сопоставляющее вершинам графа управления элементы X , назовем разметкой. Поточное распространение отношений равенства и порядка вводит аналогичные отношения на множестве разметок.

Функцией перехода назовем отображение F , которое переводит одну разметку в другую. Разметку μ_s назовем *неподвижной точкой* отображения F тогда и только тогда, когда $F(\mu_s) = \mu_s$.

Неформально говоря, разметка представляет собой некоторый набор потоковых утверждений (т.е. утверждений о свойствах потоков данных) для каждой вершины графа. Решение задачи в этом случае также может быть представлено с помощью такой разметки, а процесс решения задачи анализа потоков данных может быть описан как последовательное уточнение разметки, отталкиваясь от некоторой начальной.

Итеративный алгоритм

$(X, <)$ - множество конечной высоты N :

$$\forall \{x_1, x_2, \dots\}, x_i \in X, x_i < x_{i+1} \quad \forall k, l \ (k > N) \& (l > N) \Rightarrow x_k = x_l$$

F - функция перехода:

$$\forall \mu \ F(\mu) \geq \mu$$

Итеративный алгоритм:

μ_0 - начальная разметка

$$\mu_c = \mu_0$$

while $(F(\mu_c) \neq \mu_c)$ do $\mu_c = F(\mu_c)$;

Итеративный алгоритм

Основной проблемой описанного выше подхода является проблема остановки алгоритма. Действительно, в какой момент процесс уточнения разметки должен прекратиться? Очевидно, в тот момент, когда получено решение задачи анализа потоков данных. Однако поскольку решение задачи неизвестно, то и воспользоваться этим наблюдением напрямую оказывается невозможным. Поэтому для определения завершаемости алгоритма используется другой принцип – принцип достижения неподвижной точки.

Частично-упорядоченное множество X будем называть множеством конечной высоты N тогда и только тогда, когда длины всех строго возрастающих последовательностей элементов X ограничены N . Это означает, что для произвольной возрастающей последовательности начиная с некоторого места все элементы становятся одинаковыми.

Рассмотрим теперь функция перехода F , удовлетворяющую соотношению $F(\mu) \geq \mu$ для произвольной разметки μ . Понятно, что при таком условии при итерировании F начиная с некоторого места будет достигнута ее неподвижная точка. Множество X и функция перехода F подбираются таким образом, чтобы эта неподвижная точка являлась решением задачи анализа потоков данных.

На слайде приведена схема итеративного алгоритма анализа потоков данных.

Далее мы более детально рассмотрим возможную природу множества фактов X , множества разметок и преобразователей F .

Полурешетки

L - множество с операцией \wedge :

- $x \wedge x = x$
- $x \wedge y = y \wedge x$
- $x \wedge (y \wedge z) = (x \wedge y) \wedge z$

Индукцированное отношение порядка:

$$x \leq y \Leftrightarrow x \wedge y = x$$

Ограниченная полурешетка:

верхняя и нижняя грани:

$$\perp_L: \forall x \in L \ \perp_L \leq x$$

$$T_L: \forall x \in L \ x \leq T_L$$

Монотонные функции:

$$f: L \rightarrow L: x \leq y \Rightarrow f(x) \leq f(y)$$

Дистрибутивные функции:

$$f: L \rightarrow L: f(x \wedge y) = f(x) \wedge f(y)$$

Полурешетки

Полурешеткой называется множество, снабженное идемпотентной, коммутативной и ассоциативной операцией \wedge (определение свойств этой операции приведено на слайде). При наличии такой операции естественным образом индуцируется отношение частичного порядка.

Полурешетка L называется *ограниченной* тогда и только тогда, когда в ней существуют наибольший T_L и наименьший \perp_L элементы.

Функция f называется монотонной, если она сохраняет отношение порядка и дистрибутивной, если она является гомоморфизмом относительно полурешеточной операции. Можно показать, что дистрибутивная функция всегда монотонна.

Неподвижные точки монотонной функции

L - ограниченная полурешетка конечной высоты

$f: L \rightarrow L$ - монотонная функция \Rightarrow

- $\exists x \in L : f(x) = x$
- $L_f = \{x \in L : f(x) = x\}$ - ограниченная полурешетка конечной высоты
- $T_{L_f} = f^n(\perp)$ - наименьшая неподвижная точка, где
$$f^0(x) = x$$
$$f^n(x) = f(f^{n-1}(x))$$

Неподвижные точки монотонной функции

Пусть L – ограниченная полурешетка конечной высоты, f – монотонная функция. Можно показать тогда, что

- функция f обладает хотя бы одной неподвижной точкой
- множество всех неподвижных точек f является ограниченной полурешеткой конечной высоты
- наименьшая неподвижная точка f может быть получена итерированием функции f начиная с наименьшего элемента L

Пример

$$A = \{a, b, c, d, \dots, z\}$$

$L = 2^A$ - ограниченная полурешетка конечной высоты $|A|$

$$\bullet x \wedge y = x \cap y$$

$$\bullet T_L = A$$

$$\bullet \perp_L = \emptyset$$

$f(x) = x \cup \{a\}$ - монотонная функция

$$\bullet T_{L_f} = A$$

$$\bullet \perp_{L_f} = \{a\}$$

$$\bullet L_f = \{x : a \in x\}$$

Пример

В качестве примера ограниченной полурешетки конечной высоты рассмотрим множество всех подмножеств букв латинского алфавита с операцией пересечения. Понятно, что данная полурешетка является ограниченной – в качестве наибольшего элемента выступает множество всех букв, в качестве наименьшего – пустое множество. Так как множество всех букв конечно, то высота данной полурешетки также будет конечной.

Рассмотрим функцию, которая к своему аргументу добавляет букву a . Легко видеть, что эта функция является монотонной. Наименьшей неподвижной точкой этой функции является множество, состоящее из единственной буквы a , множество всех ее неподвижных точек есть множество подмножеств букв, содержащих букву a .

Произведение полурешеток

L_1, L_2, \dots, L_k - ограниченные полурешетки конечной высоты \Rightarrow

$L = L_1 \times L_2 \times \dots \times L_k = \{ \langle x_1, x_2, \dots, x_k \rangle, x_i \in L_i \}$ - ограниченная полурешетка конечной высоты:

- $\langle x_1, x_2, \dots, x_k \rangle \wedge \langle y_1, y_2, \dots, y_k \rangle = \langle x_1 \wedge y_1, x_2 \wedge y_2, \dots, x_k \wedge y_k \rangle$
- $\langle x_1, x_2, \dots, x_k \rangle \leq \langle y_1, y_2, \dots, y_k \rangle \Leftrightarrow (x_1 \leq y_1) \& (x_2 \leq y_2) \& \dots \& (x_k \leq y_k)$
- $T_L = \langle T_{L_1}, T_{L_2}, \dots, T_{L_k} \rangle$
- $\perp_L = \langle \perp_{L_1}, \perp_{L_2}, \dots, \perp_{L_k} \rangle$

f_1, f_2, \dots, f_k - монотонные функции на $L_1, L_2, \dots, L_k \Rightarrow$

$f(\langle x_1, x_2, \dots, x_k \rangle) = \langle f(x_1), f(x_2), \dots, f(x_k) \rangle$ - монотонная функция на L

Произведение полурешеток

Для дальнейшего изложения нам потребуется ввести операцию декартова произведения полурешеток.

Если L_1, L_2, \dots, L_k — ограниченные полурешетки конечной высоты, то такую же структуру можно ввести и на декартовом произведении этих полурешеток, определяя соответствующие понятия (операцию, наибольший и наименьший элементы) покомпонентно.

Набор монотонных функций f_1, f_2, \dots, f_k соответственно на полурешетках L_1, L_2, \dots, L_k аналогичным образом индуцирует монотонную функцию на их декартовом произведении.

Формализация задачи анализа потоков данных

- граф потока управления G
- ограниченная полурешетка конечной высоты L
- $\forall v \in V f_v : L \rightarrow L$ - монотонная функция перехода
- разметка $before : V \rightarrow L$
- разметка $after : V \rightarrow L$
- наименьшее решение системы уравнений

$$\left\{ \begin{array}{l} before(v) = \bigwedge_{w \in pred(v)} after(w) \\ after(v) = f_v(before(v)) \end{array} \right\}_{v \in V} \quad (*)$$

Формализация задачи анализа потоков данных

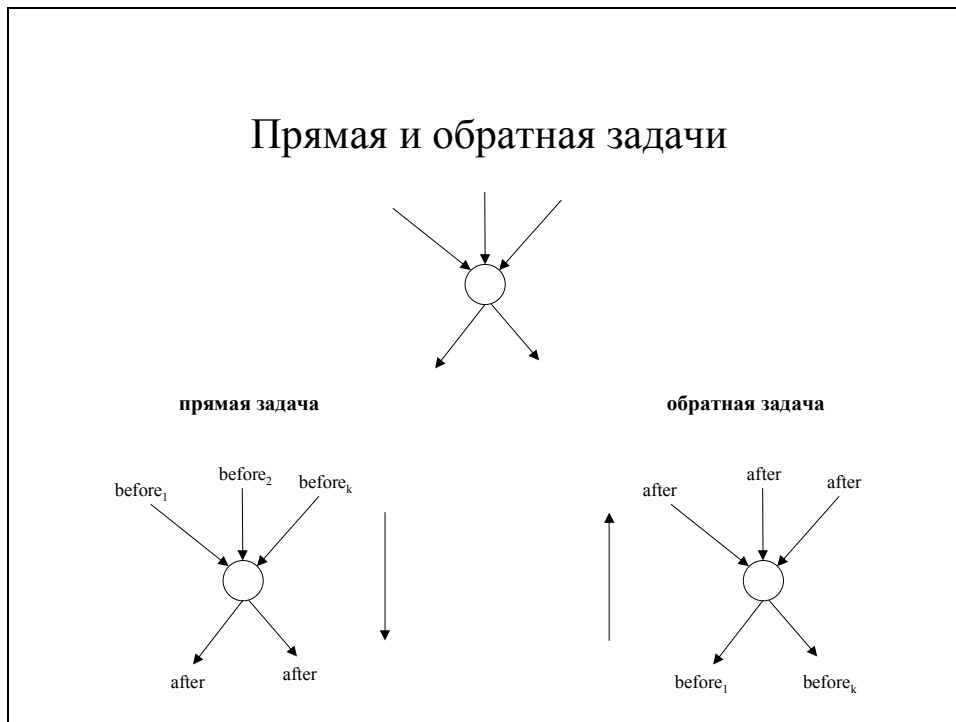
Теперь мы готовы к тому, чтобы дать формальную постановку задаче анализа потоков данных.

Пусть есть ограниченная полурешетка конечной высоты L , граф потока управления G и набор монотонных на L потоковых функций f_v для каждой вершины v графа G .

Тогда решением задачи анализа потоков данных называется пара наименьших разметок $before, after$, являющихся решением системы уравнений $(*)$, приведенной на слайде.

Неформально говоря, полурешетка L представляет собой множество потоковых фактов, разметка $before$ описывает решение задачи анализа потоков данных до исполнения операторов в данной вершине графа, а разметка $after$ – после. Решеточная операция описывает получение общей части нескольких решений. Систему уравнений можно интерпретировать следующим образом: для каждой вершины решением задачи до нее является общая часть решений всех предшественников данной вершины, а после нее – применение к этой общей части потоковой функции, ассоциированной с данной вершиной.

Прямая и обратная задачи



Прямая и обратная задачи

Приведенное выше определение описывает так называемую *прямую* задачу. Она характеризуется тем, что фактически разметка *before* ассоциируется с входящими ребрами вершины, а разметка *after* — с исходящими. Таким образом, потоковая информация как бы "перемещается" сверху-вниз.

Естественным образом возникает симметричное определение, при котором разметка *before* ассоциируется с исходящими ребрами, а разметка *after* — с входящими. При этом потоковая информация распространяется снизу-вверх. Видно, что обратная задача превращается в прямую при изменении направлений всех ребер на противоположные.

Далее мы рассмотрим примеры постановки конкретных задач анализа потоков данных, используя описанный выше формализм.

Достижимые определения

A - множество присваиваний

$L=2^A$ - полурешетка фактов:

$$\bullet x \wedge y = x \cup y$$

$$\bullet T_L = A$$

$$\bullet \perp_L = \emptyset$$

Функции перехода:

$$\forall v \in V \quad f_v(x) = x \cup \{\text{множество присваиваний в } v\}$$

Начальные разметки:

$$\forall v \in V \quad before(v) = after(v) = \emptyset$$

Достижимые определения

Достижимые определения являются прямой задачей. Будем считать, что каждая вершина графа содержит не более одного присваивания произвольной переменной.

В качестве полурешетки потоковых фактов фиксируется множество подмножеств присваиваний с операцией объединения. Наибольшим и наименьшим элементами данной полурешетки являются соответственно множество всех присваиваний и пустое множество. Очевидно, данная полурешетка имеет конечную высоту.

В качестве потоковых функций для каждой вершины графа определим функцию, которая добавляет к своему аргументу множество всех присваиваний в данной вершине.

Наконец, начальными разметками являются разметки, сопоставляющие вершинам графа наименьшие элементы полурешетки.

Живые переменные

A - множество переменных

$L=2^A$ - полурешетка фактов:

- $x \wedge y = x \cup y$
- $T_L = A$
- $\perp_L = \emptyset$

Функции перехода:

$$\forall v \in V \quad f_v(x) = (x \setminus D) \cup U, \text{ где}$$

D - множество переменных в левой части присваиваний в v

U - множество используемых переменных в v

Начальные разметки:

$$\forall v \in V \quad before(v) = after(v) = \emptyset$$

Живые переменные

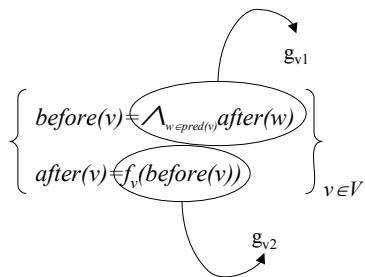
Живые переменные – это обратная задача. В качестве полурешетки потоковых фактов выбирается множество подмножеств переменных с операцией объединения, наибольший и наименьший элементы полурешетки очевидны, так же как и факт конечности высоты.

Для произвольной вершины v определим множество D_v как совокупность всех переменных, встречающихся в левых частях всех присваиваний в v , множество U_v как совокупность всех переменных, имеющих иные вхождения в операторы v . Определим для каждой вершины v потоковую функцию

$$f_v(x) = (x \setminus D_v) \cup U_v$$

В качестве начальной разметки также избирается разметка исходного графа наименьшим элементом полурешетки.

Решение задачи анализа потоков данных (1)



Пример:

$$before(v) = after(w) \wedge after(u) \Rightarrow g_{v1}(a, b) = a \wedge b$$

$$after(v) = f_v(before(v)) \Rightarrow g_{v2}(a) = f_v(a)$$

Решение задачи анализа потоков данных (1)

Для описания решения задачи анализа потоков данных рассмотрим вновь систему (*). Для каждой пары уравнений системы введем пару вспомогательных функций g_{v1} , g_{v2} , каждая из которых вычисляет значение правой части соответствующего уравнения (см. пример на слайде).

Можно показать, что полученные таким образом функции являются монотонными.

Решение задачи анализа потоков данных (2)

Функция перехода: $F : L^{2 \times |V|} \rightarrow L^{2 \times |V|}$

$$F(<before_1, after_1, before_2, after_2, \dots, before_{|V|}, after_{|V|}>) = \\ <g_{11}(\dots), g_{12}(\dots), \dots, g_{|V|1}(), g_{|V|2}(\dots)>$$

Свойства:

- F - монотонна на $L^{2 \times |V|}$
- произвольная неподвижная точка F является решением системы (*)
- $\exists n: F^n(\perp_{L^{2 \times |V|}}, \perp_{L^{2 \times |V|}}, \dots, \perp_{L^{2 \times |V|}})$ - наименьшая неподвижная точка F

Решение задачи анализа потоков данных (2)

Введем в рассмотрение полурешетку, представляющую собой декартову степень $2^{|V|}$ исходной полурешетки L и введем в рассмотрение функцию F на ней. Данная функция принимает на вход элемент $<before_1, after_1, \dots, before_{|V|}, after_{|V|}>$ и возвращает элемент, полученный применением функций, построенных на предыдущем шаге, к соответствующим аргументам. Иными словами, если, например, для вершины 10 графа потока управления предшественниками являются вершины 2 и 3 , то двадцатым элементом возвращаемого F значения будет $g_{10,2}(after_2, after_3)$.

Легко видеть, что функция F является монотонной. Кроме того, можно показать, что ее произвольная неподвижная точка является решением системы уравнений (*), поскольку произвольный элемент решетки $L^{2 \times |V|}$ определяет пару разметок $before, after$.

Наконец, итерирование функции F , начиная с наименьшего элемента, дает в конце концов наименьшую неподвижную точку и, следовательно, искомое решение задачи анализа потоков данных.

Свойства итеративного подхода

- Нахождения точного решения задачи анализа потоков данных неразрешимо
- Если все f_v - дистрибутивны, то наименьшая неподвижная точка F есть точное решение задачи анализа потоков данных

Свойства итеративного подхода

Может быть показано, что нахождение точного решения задачи анализа потоков данных (т.е. наименьшего общего набора свойства при исполнении по *всем* путям до данной вершины) неразрешимо. Таким образом, итеративный подход дает только приближенное решение.

Однако доказано, что в случае, когда все потоковые функции не просто монотонны, но и дистрибутивны, итеративное решение является точным.

Алгоритм с рабочим списком

<pre>void Traverse (W: list<Vertex>) { if (W is empty) return; else { Vertex v = any of W; W = W \ {v}; after = $f_v(\bigwedge_{w \in \text{pred}(v)} \text{after}(w))$; if (after \neq after (v)) { after (v) = after; W = W \cup succ(v); } } Traverse (W); }</pre>	<pre>void DFA (G: CFG, L: Semilattice) { for $\forall v \in V$ do before(v)=after(v)=\perp_L; Traverse ({start}); }</pre>
--	--

Алгоритм с рабочим списком

Приведенный способ решения задачи анализа потоков данных может быть реализован с помощью алгоритма, приведенного на иллюстрации. Заметим, что технически нет необходимости в явном виде представлять декартову степень полурешетки и преобразователь F .

Очевидно, что достаточно привести запись алгоритма для прямой задачи.

Алгоритм производит итеративное перевычисление разметок, используя рабочий список вершин. Главным свойством этого списка является то, что он состоит из вершин, для предшественников которых значение разметок было изменено на предыдущем шаге. Таким образом, опустошение списка свидетельствует о том, что достигнута неподвижная точка.

Алгоритм начинает работу на начальной разметке и рабочем списке, состоящем из единственной вершины *start* (для обратной задачи – *stop*). Извлекая очередную вершину из рабочего списка, алгоритм вычисляет общую часть решения задачи по всем своим предшественникам и применяет к ней потоковую функцию, ассоциированную с данной вершиной. Если полученное значение отличается от текущего значения разметки *after* для данной вершины, то все ее наследники добавляются в рабочий список.

Пример: достижимые определения (1)

```

int F (int a, int b)
{
  int g = a, m = b;

  if (a < b) {g = b; m = a;}

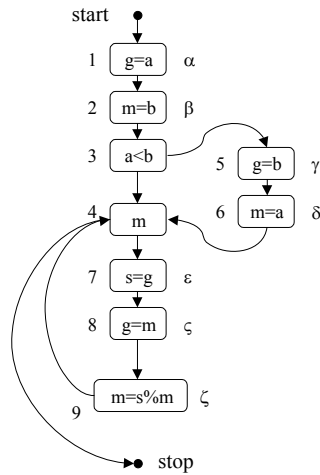
  while (m)
  {
    int s = g;

    g = m;

    m = s % m;
  }

  return g;
}

```

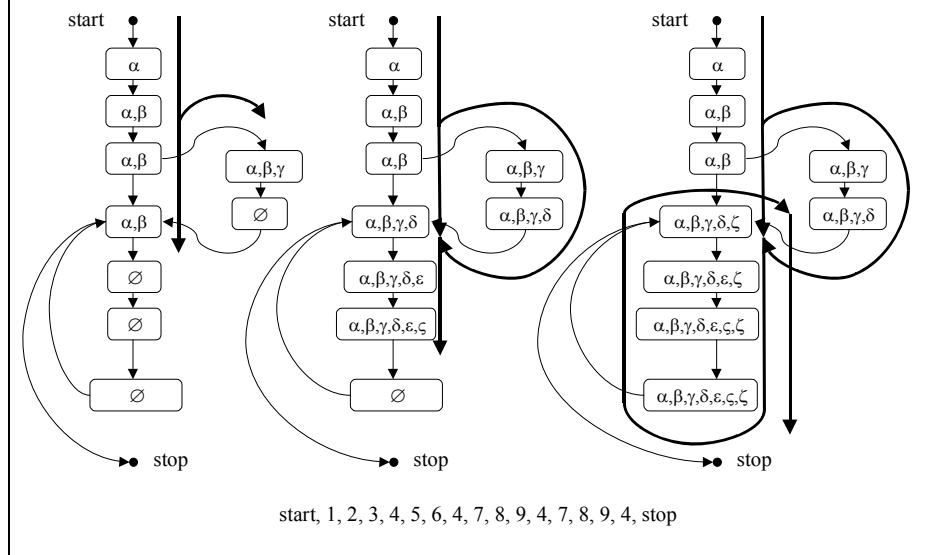


$f_1(x) = x \cup \{\alpha\}$
 $f_2(x) = x \cup \{\beta\}$
 $f_3(x) = x$
 $f_4(x) = x$
 $f_5(x) = x \cup \{\gamma\}$
 $f_6(x) = x \cup \{\delta\}$
 $f_7(x) = x \cup \{\epsilon\}$
 $f_8(x) = x \cup \{\zeta\}$
 $f_9(x) = x \cup \{\zeta\}$

Пример: достижимые определения (1)

Продemonстрируем работу алгоритма на примере задачи о достижимых определениях. На иллюстрации слева показана исходная программа, в центре – ее граф потока управления. Слева от узлов указаны их номера, справа – греческими буквами обозначены присваивания. Набор потоковых функций показан в правой части иллюстрации.

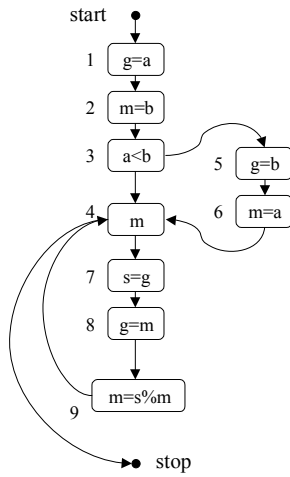
Пример: достижимые определения (2)



Пример: достижимые определения (2)

На иллюстрации показано несколько состояний разметки в процессе работы алгоритма. Жирными стрелками обозначен порядок обход графа, внутри вершин показана разметка *after*. Видно, что при первом входе в вершину 4 вершины 6 и 9 еще необработанны (правая часть иллюстрации). После первого прохода по вершинам 7, 8 и 9 неподвижная точка еще не достигнута (средняя часть иллюстрации), что требует еще одного прохода по фрагменту пути 4, 7, 8. Окончательная разметка показана в правой части иллюстрации. Возможный порядок посещения вершин при работе алгоритма показан внизу иллюстрации.

Пример: живые переменные (1)



$$D_1 = \{g\}, U_1 = \{a\}, f_1(x) = x \setminus \{g\} \cup \{a\}$$

$$D_2 = \{m\}, U_2 = \{b\}, f_2(x) = x \setminus \{m\} \cup \{b\}$$

$$D_3 = \emptyset, U_3 = \{a, b\}, f_3(x) = x \setminus \{a, b\}$$

$$D_4 = \emptyset, U_4 = \{m\}, f_4(x) = x \setminus \{m\}$$

$$D_5 = \{g\}, U_5 = \{b\}, f_5(x) = x \setminus \{g\} \cup \{b\}$$

$$D_6 = \{m\}, U_6 = \{a\}, f_6(x) = x \setminus \{m\} \cup \{a\}$$

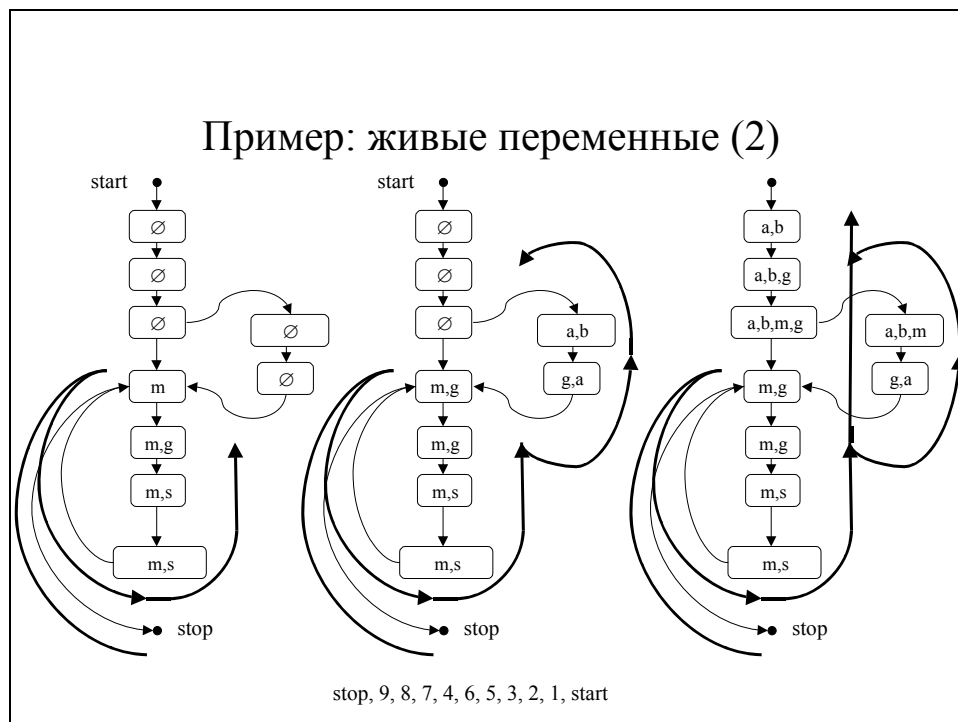
$$D_7 = \{s\}, U_7 = \{g\}, f_7(x) = x \setminus \{s\} \cup \{g\}$$

$$D_8 = \{g\}, U_8 = \{m\}, f_8(x) = x \setminus \{g\} \cup \{m\}$$

$$D_9 = \{m\}, U_9 = \{s, m\}, f_9(x) = x \setminus \{s\}$$

Пример: живые переменные (1)

В качестве примера работы алгоритма для обратной задачи рассмотрим задачу о живых переменных для той же программы. Граф в тех же обозначениях и множество потоковых функций показаны на слайде.



Пример: живые переменные (2)

Последовательность работы алгоритма приведена на иллюстрации. Как и в предыдущем случае, последовательность обработки вершин обозначена жирными стрелками.

Заключение

Шаги, необходимые для решения задачи анализа потока данных с помощью итеративного подхода:

- формализовать решение задачи с помощью подходящей полурешетки
- описать преобразование потоков данных при проходе через вершину графа управления с помощью монотонной, а лучше дистрибутивной функции
- применить итеративный алгоритм.

Заключение

В заключение опишем еще раз последовательность шагов, которую надо осуществить для решения задачи анализа потоков данных итеративным способом.

Прежде всего, необходимо формализовать множество фактов и решение задачи анализа потоков данных, придумав подходящую полурешетку.

Затем, необходимо описать преобразование множества потоковых фактов при прохождении через вершину графа с помощью монотонных, а еще лучше дистрибутивных функций.

Наконец, применить итеративный алгоритм (в прямой или обратной модификации) для получения неподвижной точки.

Литература к лекции

- А. Ахо, Р. Сети, Дж. Ульман. "Компиляторы: принципы, технологии и инструменты", М.: "Вильямс", 2001. 768 с.
- Steven S. Muchnik "Advanced Compiler Design And Implementation". Morgan Kaufmann Publishers, July 1997, 880 pp.
- В.Н.Касьянов "Оптимизирующие преобразования программ", М., "Наука", 1988. 336 с.

Литература к лекции

- А. Ахо, Р. Сети, Дж. Ульман "Компиляторы: принципы, технологии и инструменты", М.: "Вильямс", 2001. 768 с.
- Steven S. Muchnik "Advanced Compiler Design And Implementation", Morgan Kaufmann Publishers, July 1997. 880 pp.
- В.Н. Касьянов "Оптимизирующие преобразования программ", М., "Наука", 1988. 336 с.