О ФОРМАЛЬНЫХ ПРЕОБРАЗОВАНИЯХ АЛГОРИТМОВ

В.М.Глушков
Киев, СССР

Современные формы представления алгоритмов в виде программ в алгоритмических языках можно рассматривать как естественное развитие формульного аппарата классической математики, используемого для конструктивного представления математических объектов. Это приводит к идее разработки методов формальных преобразований алгоритмов, рассматриваемых как алгебраические выражения. С этой целью автором в 1965 г. была предложена алгебра алгоритмов и доказана теорема анализа, показывающая возможность представления в этой алгебре произвольных программ [1] . В этой же работе был рассмотрен пример построения эффективного алгоритма путем преобразования начального описания с помощью соотношений в алгебре алгоритмов.

В дальнейшем продолжалось изучение соотношений в алгебре алгоритмов. Она использовалась при создании ряда систем автоматизации проектирования схемного и программного оборудования ЭВМ. В работе [2] были выведены формулы алгебры алгоритмов, которые могут быть использованы для ускорения итерации монотонных операторов. Эти формулы обобщают метод, использованный в [1] для ускорения умножения, и позволяют использовать его для ускорения других алгоритмов. Рассмотрим это обобщение.

Пусть рассматриваемая алгебра алгоритмов порождается элементарными операторами и условиями, действующими на информационном множестве B . Как известно, эта алгебра является двухосновной, составленной из двух компонент: алгебры операторов и алгебры условий. Элементами первой алгебры являются операторы, т.е. частичные преобразования множества B , элементами второй алгебры – условия, т.е. частичные функции, заданные на B и принимающие значения 0,1. Операциями алгебры операторов являются произведение, α-дизъюнкция и α-итера-

ция. В алгебре условий определены булевские операции и операция умножения оператора на условие. Эту операцию мы будем обозначать α^P (α после P).

Оператор y называется монотонным относительно условия α, если из $\alpha(b) = I$ следует, что $\alpha(y(b)) = I$. Будем предполагать, что множество B состояний информационной среды состоит из двух компонент C и C', т.е. $B = C \times C'$. Операторы P и Q называются C-эквивалентными, если они имеют одинаковые области определения и для любого состояния $b = (c,c')$ из $P(c,c') = (c_1,c_1')$ и $Q(c,c') = (c_2,c_2')$ следует $c_1 = c_2$. Иными словами, C-эквивалентность означает одинаковые действия на компоненте C. Будем говорить, что оператор y (условие α) действует на компоненте C, если существует частичная функция φ из C в C (из C в $\{0, I\}$) такая, что для любых $c \in C$, $c' \in C'$ $y(c, c') = (\varphi(c), c')$ $(\alpha(c, c') = \varphi(c))$.

Аналогично можно говорить о действии на компоненте C'.

Основной результат работы [2] может быть сформулирован следующим образом. Пусть оператор y и условие α действуют на компоненте C, операторы z_0 и u действуют на компоненте C' информационной среды $B = C \times C'$, а в полугруппе операторов алгебры алгоритмов, действующих на B, выполняются соотношения:

$$z_0 z = z_0 y;$$

$$z_0 u^{m+1} z = z_0 u^m z^2 u, \quad m = 0,1,\ldots,$$

$$u z_0 = z_0.$$

Тогда, если оператор y монотонен относительно условия α, то следующие операторы C-эквивалентны:

$$\{y\}_{\alpha}; \tag{1}$$

$$\{z_0 \{u\}_{\alpha^{z^2}} z\}_{\alpha}; \tag{2}$$

$$z_0 \{ \underset{\alpha}{\{u\}} \underset{\beta}{z} \}, \text{ где } \beta = \alpha^{\alpha^{\{z^2\}}_{z}}, \tag{3}$$

$$z_0 \{ \underset{\alpha}{(\varepsilon \vee z)u} \}, \ \beta - \text{ то же, что в (3)} \tag{4}$$

Если кроме того, в полугруппе операторов есть оператор u^{-1}, действующий на компоненте C и удовлетворяющий соотношениям

$$z_0 u^{m+1} u^{-1} = z_0 u^m, \ m = 0,1,\ldots,$$

а также оператор y^{-1}, действующий на C и такой, что $yy^{-1} = \varepsilon$, то операторы (I)-(4) C-эквивалентны оператору

$$z_0 \underset{\alpha^z}{\{u\}} \{ \underset{\alpha^{zy^{-1}}}{\alpha} u^{-1} \} z\}. \tag{5}$$

Применение метода ускорения итераций монотонных операторов происходит обычно в условиях, когда итерация определена для алгебры алгоритмов, действующей на информационной среде C. Для построения эффективной программы C расширяется до множества $C \times C'$, где C' выбирается таким образом, чтобы можно было определить операторы z_0, u, z, а если возможно, то u^{-1} и y^{-1}. После этого выбирается одна из программ (2)-(5) и реализуется в используемом алгоритмическом языке. Для перехода к обычной программе достаточно реализовать проверку условий. Если простая итерация работает на некотором состоянии информационной среды n раз, то программа, построенная по формуле (2), работает время, пропорциональное $(\log n)^2$, а программы (3)-(5) - время, пропорциональное $\log n$.

Методы формального преобразования алгоритмов широко используются в развиваемой в Институте кибернетики АН УССР технологии проектирования, получившей название метода формализованных технических заданий [3,4] . Рассмотрим несколько примеров формальных преобразований алгоритмов в процессе их проектирования этим методом, выполненных в последнее время.

Первый пример возьмем из работы [5] . Рассмотрим задачу

вычисления числа компонент связанности неориентированного графа $\Gamma = (G, \rho)$, определенного множеством вершин G и симметричным отношением смежности $\rho \subset G^2$. Рефлексивное и транзитивное замыкание отношения ρ дает отношение достижимости $\hat{\rho}$: g и g' достижимы, если $g = g'$ или существует последовательность $g = g_1, g_2, \ldots, g_m = g'$ $(m \geq 1)$ вершин такая, что g_i и g_{i+1} смежны для $i = 1, \ldots, m-1$. Отношение достижимости является эквивалентностью. Классы этого отношения эквивалентности будем называть компонентами связности. Компоненту связности, содержащую вершину g, обозначим через $\hat{\rho}(g)$.

Следующая программа описывает в теоретико-множественных терминах простой алгоритм решения этой задачи:

```
BEGIN.
  k:=0.
  FOR ALL g ∈ G DO φ(g):=0.
LOOP BEGIN.
  FIND g ∈ G SUCH THAT φ(g) = 0.
  IF NOT THEN GO OUT.
  k:=k + 1.
  FOR ALL h ∈ ρ̂(g) DO  φ(h):=k.
END LOOP.
END.
```

Корректность этой программы легко может быть доказана обычными методами. Представив рассматриваемую программу в виде

```
BEGIN.
  k:=0.
  P.
LOOP BEGIN.
  Q.
  k:=k + 1.
  R.
END LOOP.
END.
```

и выполняя последовательное уточнение операторов P, Q, R, получим программу в АЛГОЛе. Для построения этой программы выберем следующий способ представления графа. Пусть G—целочисленный массив размером [1:n, 1:m], где n—число вершин графа Г, m - не меньше, чем максимальное число вершин, смежных с любой данной вершиной графа Г. Массив G представляет граф Г, если G[i,j] $\neq$ 0, тогда и только тогда, когда вершины g_i и $g_{G[i,j]}$ смежны, i = 1, ... , n, j = 1,...,m, G = $\{g_1, \ldots, g_n\}$. Функцию φ реализуем с помощью целочисленного массива FI[1:n], полагая $\varphi(g_i)$ = = FI[i] для i = 1,...,n. Программа в языке АЛГОЛ имеет следующий вид:

```
    PROCEDURE COMP (G, M, N, K); INTEGER ARRAY G;
INTEGER N, M, K; VALUE M, N;
BEGIN INTEGER ARRAY FI[1:N],INTEGER I,J,S;BOOLEAN CHANGE;
P[  K: = 0;
 FOR I: = 1 STEP 1 UNTIL N DO FI[I]: = 0;
Q[LFOR I: = 1 STEP 1 UNTIL N DO IF[I] = 0 THEN
 BEGIN S: = I; GO TO FOUND END; GO TO FINE;
FOUND:K: = K + 1;
```

```
    FI[S]: = K;
REPEAT:CHANGE: = FALSE;
    FOR I: = 1 STEP 1 UNTIL N DOIFFI[I] = K THEN
R   FOR J: = 1 STEP 1 UNTIL M DOIF G[I,J] ≠ 0 THEN
        IF FI[G[I,J]] ≠ K THEN BEGIN FI[6[I,J]]: = K;
        CHANGE: = TRUE END;
        IF CHANGE THEN GO TO REPEAT; GO TOL;
FINE : END.
```

Первая программа рассматривается как формализованное техническое задание, реализацией которого является вторая программа. Полученная программа неудовлетворительна, поскольку работает время, пропорциональное mn^2. Для того, чтобы сократить время работы этой программы, вернемся на теоретико-множественный уровень, сохранив представление данных.

```
BEGIN.
    k: = 0.
[FOR ALL i = 1,...,n·DO FI[i]: = 0.
LOOP BEGIN.
Q   FIND s SUCH THAT FI[s] = 0.
    IF NOT THEN GO OUT.
    k: = k + 1.
    FI[s]: = k.
    DO UNTIL   CHANGE
    BEGIN. CHANGE: = FALSE.
      FOR ALL i = 1,...,n SUCH THAT FI[i] = k DO
R     FOR ALL j = 1,...,m SUCH THAT G[i,j] ≠ 0
          AND FI[G[i,j]] ≠ k DO
      (FI[G[i,j]]: = k. CHANGE: = TRUE).
    END UNTIL.
END LOOP.
END.
```

Введем новые структуры данных: множество $V_0 \subset \{1,\ldots,n\}$ и множество $W \subset \{1,\ldots,n\}$. Добавим в рассматриваемую программу вычисления над этими множествами, построив их таким образом, чтобы при каждом прохождении внешнего цикла выполнялось условие $V_0 = \{1 \le i \le n \mid FI[i] = 0\}$, а при каждом прохождении внутреннего цикла – условие $FI[i] = k$ и существовало $1 \le j \le m$ такое, что $G[i,j] \neq 0$ и $FI[G[i,j]] \neq k \Rightarrow i \in W$". Теперь оператор Q можно заменить оператором

IF $V_0 = \emptyset$ THEN GO OUT ELSE GET s FROM V_0,

а заголовок цикла по i – заголовком FOR ALL $i \in W$ DO. Выполнение оператора GET X FROM A сопровождается удалением элемента X из A, а оператор FOR ALL $i \in W$ DO T эквивалентен следующей последовательности операторов:

```
WHILE  W ≠ ∅  DO BEGIN.
    GET i FROM W.
    T.
END WHILE.
```

Для того, чтобы перед выполнением этой последовательности выполнялось условие для множества W, достаточно в операторе T добавлять к этому множеству индекс всякой новой вершины в момент, когда она впервые отмечается числом k. После соответствующих преобразований станет ясно, что при втором прохождении внутреннего цикла множество W будет пустым и скобки цикла можно опустить. Мы можем также удалить переменную CHANGE. Измененная программа имеет следующий вид:

```
BEGIN.
  k: = 0.
  FOR ALL i = 1,...,n DO FI[i]: = 0.
< V0: = {1,...,n}>
WHILE V0 ≠ ∅ DO BEGIN.
  GET s FROM V0.
  k: = k + 1. FI[s]: = k.
< W: = {s}.>
  FOR ALL i ∈ W DO
  FOR ALL j=1,...,m SUCH THAT G[i,j]≠0 AND FI[G[i,j]]=0 DO
  (FI[G[i,j]]: = k.< DELETE G[i,j] FROM V0.
  INCLUDE G[i,j] IN W>. )
END WHILE.
END.
```

В этой программе вставленные операторы взяты в скобки < > . Полученная программа будет работать линейное время, если удаление и добавление элементов множеств V_0 и W осуществляется за ограниченное время. Этого легко добиться, реализовав представление с помощью списков. Аналогичным образом можно было бы получить экономный алгоритм подсчета числа компонент связности для ориентированных графов, с временной оценкой такой же, как в известной работе Тарьяна [6] . Подобным же методом в [5] осуществлено получение алгоритма Хопкрофта [7] приведения автомата за время $n \log n$ из классического алгоритма приведения, имеющего временную характеристику n^2.

Следующий пример также относится к теоретико-множествен-

ному программированию и связан с реализацией рекурсивных определений.

Пусть Q есть n-местное отношение, заданное следующим рекурсивным определением:

$$P(x_1,\ldots,x_n) \Rightarrow Q(x_1,\ldots,x_n);$$

$$Q(x_1,\ldots,x_n) \text{ и } R(z_1,\ldots,z_m) \Rightarrow Q(y_1,\ldots,y_n);$$

Q есть наименьшее отношение, удовлетворяющее первым двум условиям для всех $x_1,\ldots,x_n$ и $u_1,\ldots,u_k$ таких, что $S(u_1,\ldots,u_k)$.

Здесь R и S, заданные отношения, $z_1,\ldots,z_m$, $y_1,\ldots,y_n$ - алгебраические выражения, зависящие от переменных $x_1,\ldots,x_n$, $u_1,\ldots,u_k$.

От этого определения легко перейти к теоретико-множественной программе, которая строит множество Q, если все отношения, используемые в определении, конечны. Программа имеет следующий вид:

```
BEGIN. Q: = ∅. Q_0: =  .
 LOOP BEGIN. Q_1: = ∅.
    FOR ALL (x_1,...,x_n) ∈ Q_0 DO
    FOR ALL (u_1,...,u_k)∈ S SUCH THAT R(z_1,...,z_m) DO
    IF (y_1,...,y_n) ∉ Q ∪ Q_0 ∪ Q_1 THEN INCLUDE (y_1,...,y_n)
    IN Q_1.
    Q: = Q ∪ Q_0.
    IF Q_1 = ∅ THEN GO OUT.
    Q_0: = Q_1.
 END LOOP.
END.
```

Обычно такая программа работает неэффективно, перебирая лишние элементы множеств P, R и S по нескольку раз. Поэтому ее следует оптимизировать, пользуясь конкретными свойствами множеств P, R и S.

Рассмотрим хорошо известную задачу определения существенных переменных в состоянии схемы программы – одну из основных задач анализа потоков данных в программах. Рассмотрим необходимые определения. Схема программы над памятью R – это множество состояний схемы вместе с множеством T переходов, каждый из которых представляет собой четверку (a, u, y, a'), где a – состояние схемы, u – условие перехода, y – оператор, выполняемый во время этого перехода, a' – состояние после этого перехода. Если (a, u, y, a') есть переход, то пишем $a \xrightarrow{u/y} a'$. Для каждого из операторов заданы два множества: множество используемых и множество вырабатываемых оператором переменных из R, а для каждого условия задано множество переменных, которое в нем используется. Пусть в схеме из a в a' есть последовательность переходов $p = t_1 \ldots t_n$ такая, что $t_i = (a_i, u_i, y_i, a_i')$, $a_1 = a$, $a_n = a'$, $a_{i+1} = a_i'$, $i = 1, \ldots, n-1$. Если p есть путь из a в a', то пишем $a \xrightarrow{p} a'$. Если переход $t = (a, u, y, a')$ такой, что u и y используют r, то говорим, что состояние a и переход t используют r. Переход t вырабатывает r, если y вырабатывает r. Основное определение формулируется следующим образом: переменная r существенна в $a \Leftrightarrow$ a использует r или существует путь p и состояние a' такие, что $a \xrightarrow{p} a'$, состояние a' использует r и ни один из переходов пути p не вырабатывает r. Для того, чтобы получить конструктивное определение понятия существенной переменной, рассмотрим отношение $Q \subset A \times R$, которое определяется как наименьшее отношение такое, что

1. a использует $r \Rightarrow (a, r) \in Q$;
2. $(a, r) \in Q$ и $a' \xrightarrow{p} a$ для некоторого перехода p, который не вырабатывает $r \Rightarrow (a', r) \in Q$.

Легко доказывается следующее предложение: переменная r существенна в $a \Leftrightarrow (a, r) \in Q$.

Применяя описанное выше построение, легко получим следующую программу, которая генерирует множество Q .

```
BEGIN. Q:= ∅. Q0: = {(a, r)| a USES r}.
  LOOP BEGIN. Q1 : = ∅.
      FOR ALL (a, r) ∈ Q0 DO
      FOR ALL a' ∈ A SUCH THAT FOR SOME p ∈ T
        (a' --p--> a AND p DOES NOT GENERATE r) DO
        IF (a', r) ∉ Q ∪ Q0 ∪ Q1 THEN INCLUDE (a',r) IN Q1.
      Q: = Q ∪ Q0.
      IF Q1 = ∅ THEN GO OUT.
      Q0: = Q1.
  END LOOP.
END.
```

Принимая во внимание способ представления схем программ, можно оценить эффективность реализации построенного алгоритма и выбрать подходящие способы представления множеств Q, Q_0, Q_1. Предположим, что множество A состояний схемы программы задается списком, а множество T переходов – с помощью функции T_1, заданной на A, и принимающей значение в множестве T, причем $p \in T_1(a) \Leftrightarrow p \in T$ и $a \xrightarrow{p} a'$ для некоторого $a' \in A$. Оценим время работы программы в зависимости от числа n состояний схемы A. При этом будем считать, что число переменных и число переходов в каждом состоянии ограничено, т.е. мало по сравнению с n, которое может быть сколь угодно большим. Тогда, если Q, Q_0 и Q_1 задаются обычными списками, то время работы будет пропорционально n^3. Сокращение времени

может произойти за счет ускорения условия $(a, r) \notin Q \cup Q_0 \cup Q_1$ и за счет ограничения области изменения параметра цикла. Первое может быть сделано, если множества Q, Q_0 и Q_1 представлять с помощью функций F, F_0 и F_1, заданных на A и принимающих значения в 2^R. При этом, $(a, r) \in Q \Leftrightarrow r \in F(a)$. Аналогично определяются F_0 и F_1. Для ускорения порождения элементов множеств Q_0 и Q_1 удобно использовать вспомогательные множества B_0 и B_1 такие, что $a \in B_0 \Leftrightarrow F_0(a) \neq \emptyset$, $a \in B_1 \Leftrightarrow F_1(a) \neq \emptyset$. Для того, чтобы ограничить множество значений параметра a', удобно предварительно построить функцию G, заданную на A и принимающую значения в 2^A. $a' \in G(a) \Leftrightarrow$ существует $p \in T$ такой, что $a \xrightarrow{p} a'$. Тогда цикл с параметром a' можно будет выполнять только по элементам множества $G(a)$, а не по всему множеству A. Выполнив формально все необходимые подстановки и преобразования, получим следующую программу:

```
BEGIN. B0: = ∅.
  FOR ALL a ∈ A DO
    BEGIN. F(a): = ∅.
      FOR ALL r ∈ R DO
      IF a USER r THEN INCLUDE r IN F0(a).
      IF F0(a) ≠ ∅ THEN INCLUDE a IN B0.
    END.
LOOP BEGIN.
    FOR ALL a ∈ B0 DO
    FOR ALL r ∈ F(a) DO
    FOR ALL a' ∈ G(a) DO
```

```
      IF FOR SOME p ∈ T1(a')(a' -p-> a AND p DOES NOT
      GENERATE r) THEN IF r ∈ F(a) OR F0(a) OR F1(a)
      THEN INCLUDE r IN F1(a'), a' IN B1.
   FOR ALL a ∈ B0 DO F(a): = F(a) ∪ F0(a).
   IF B1 ≠ ∅ THEN OUT.
   FOR ALL a ∈ B0 ∪ B1 DO
      IF a ∈ B0\B1 THEN F0(a): = ∅ ELSE
      IF a ∈ B1 THEN F0(a): = F1(a).
   FOR ALL a ∈ B1 DO F1(a): = ∅.
END LOOP.
END.
```

Во всех рассмотренных примерах использован один и тот же прием, который мы называем введением и удалением избыточных вычислений. Этот прием состоит в следующем. В алгоритм вводятся новые структуры данных и действия над этими структурами. Новые действия не влияют на результат работы алгоритма и первоначально являются избыточными, однако они приводят к тому, что в алгоритме появляются новые полезные соотношения между данными. Эти соотношения используются для выполнения оптимизирующих преобразований. После этих преобразований некоторые операторы становятся избыточными и убираются обычным путем. Формально метод введения и удаления избыточных вычислений можно сформулировать, как применение определенного набора формальных преобразований, которые могут быть точно сформулированы.

Методы формальных преобразований, проиллюстрированные здесь на конкретных примерах, работают и на больших программах. Например, в одной из последних работ оптимизировалась программа в языке PL/I, объемом порядка 2 тыс. операторов,

разработанная методом формализованных технических заданий. Применение метода введения и удаления избыточных вычислений позволило ускорить эту программу почти в 10 раз.

Применение метода формального преобразования к большим программам связано с большим объемом рутинной работы по анализу текстов программ и фактическому выполнению преобразований. Выполнение такой работы может быть облегчено использованием интерактивного преобразования программ в автоматизированных системах проектирования. Такие средства были реализованы в системе ПРОЕКТ автоматизации проектирования схемного и программного оборудования ЭВМ, разработанной в ИК АН УССР, и развиваются в настоящее время в системе теоретико-множественного программирования для решения задач искусственного интеллекта [8].

Литература

1. Глушков В.М. Теория автоматов и формальные преобразования микропрограмм.-Кибернетика, 1965, № 5.
2. Летичевский А.А. Об ускорении итерации монотонных операторов.- Кибернетика, 1976, № 4.
3. Глушков В.М., Капитонова Ю.В., Летичевский А.А. Теоретические основы проектирования дискретных систем.- Кибернетика, 1977, № 6.
4. Глушков В.М., Капитонова Ю.В., Летичевский А.А. О применении метода формализованных технических заданий к проектированию программ обработки структур данных.-Программирование, 1978, № 6.
5. Летичевский А.А., Годлевский А.Б. Оптимизация алгоритмов в процессе их проектирования методом формализованных технических заданий.- В сб.: Автоматизация проектирования ЭВМ и их компонентов. Киев, 1977.
6. Tarjan R.E. Depth first search and linear graph algorithms.- SIAM J. Comput. v.1, N2, 1972.
7. Hopcroft J.E. An nlogn algorithm for minimizing states in a finite automata, in Kohavi Z. PazA. (edes). Theory of machines and computations, Acad. Press, N-Y, 1971.
8. Глушков В.М., Капитонова Ю.В., Летичевский А.А. Инструментальные средства проектирования программ обработки математических текстов.-Кибернетика, 1979, № 2.