ОБ ИНДУКТИВНОМ СИНТЕЗЕ ПРОГРАММ

Я.М.Барздинь

Рига, СССР

§ I. ВВЕДЕНИЕ

Когда человек разъясняет какой-то алгоритм, он часто это делает путем описания работы алгоритма на некотором начальном куске вычислений. Например, алгоритм нахождения наибольшего общего делителя двух положительных целых чисел X и У, где $X \geqslant У$, обычно описывается следующим образом:

Обозначим X через A_1 и У через A_2. Далее

I) делим A_1 на A_2, находим остаток A_3 и смотрим, равен он 0 или нет. Если равен, то процесс обрывается и A_2 - искомый наибольший делитель. Если же $A_3 > 0$, то

2) A_2 делим на A_3 и находим остаток A_4. Если $A_4 = 0$, то процесс обрывается и A_3 - искомый наибольший общий делитель. Если $A_4 > 0$, то

3) делим A_3 на A_4 и т.д.

Такого рода описания алгоритмов мы будем называть индуктивными описаниями. Ясно, что индуктивное описание алгоритма существенно отличается от полного описания алгоритма в традиционных языках программирования. Для одного человека заданное индуктивное описание может быть достаточным, чтобы понять, как работает алгоритм в общем случае, для другого нет. Однако нам кажется, что человек, разъясняющий какой-то алгоритм, обычно чувствует, является ли заданное им индуктивное описание алгоритма полным, т.е. можно ли по этому описанию синтезировать полное описание алгоритма на каком-нибудь из традиционных языков программирования. Очевидно, не вызывает никаких сомнений, что приведенное выше индуктивное описание алгоритма нахождения наибольшего общего делителя является полным. По-видимому было бы чрезвычайно интересно понять логические основы такой полноты.

На наш взгляд индуктивное описание алгоритмов во многих

случаях является более "человеческим", чем "машинное" описания алгоритмов на традиционных языках программирования. Кроме того, человек при конструировании полных ("машинных") описаний алгоритмов часто проходит именно стадию индуктивного задания алгоритмов. Это он делает обычно в тот момент, когда выясняет для себя, как же алгоритм будет работать.

В данной работе делается попытка дать некоторые инструментальные средства и изучить возможность формального синтеза полных описаний алгоритмов (программ) по их индуктивным описаниям. Такие методы синтеза мы будем называть индуктивными.

В заключение отметим некоторую связь с другими работами. Теоретические аспекты индуктивного синтеза фактически рассматриваются в работах по теории индуктивного вывода (см. [I]). Однако в более непосредственной связи с нашими исследованиями находятся работы А.Бирмана [2,3], М.Бауэра [4] и др. по синтезу программ по примерам. Индуктивное описание алгоритма можно мыслить как символическое описание работы алгоритма на некотором начальном куске вычислений.

§ 2. Примеры индуктивных описаний алгоритмов

Сначала опишем языковые средства для задания индуктивных описаний алгоритмов. Эти средства не будут претендовать на общность в смысле практического применения, а нужны лишь для демонстрации наших основных идей.

Индуктивное описание алгоритмов будет состоять из двух частей - INPUT и DESCRIPTION (в § 5 будет введена еще третья составная часть - EXAMPLE):

INPUT: описание - входных - переменных

DESCRIPTION:

описание - работы - алгоритма - на - начальном - куске

Описание - входных - переменных имеет вид

$$\alpha_1, \alpha_2, \ldots, \alpha_n,$$

где α_i может быть

а) идентификатор - буква или буква с индексом, например, A, AI, X3; это означает скалярную переменную, которая в качестве значений может принимать десятичные числа с фиксирован-

ной точкой;

б) идентификатор [идентификатор I, ...];

это означает массив; например, A[N] - I - мерный массив (вектор), B[N,M] - 2 - мерный массив (матрица), N,M - идентификаторы границ массива, они в качестве значений могут принимать целые положительные числа.

Элементы массива в качестве значений могут принимать десятичные числа с фиксированной точкой. Элементы массива, как обычно, будем обозначать через A(I), A(7), B(2,3) и т.д. В случае 2-мерного массива будем пользоваться обозначениями вида B(I,) и B(,J) для изображения соответственно I-й строки и J-й колонки.

Если, например,дано

INPUT:S,B[N,M]

то S, B, N, M, а также элементы массива B будем называть внешними переменными.

Для описания - работы - алгоритма - на - начальном - куске мы будем использовать следующие средства:

а) Оператор присваивания

переменная ← выражение.

Объявлением внутренних переменных будет считаться их появление в каком-нибудь операторе присваивания; например, появление X(2) ← 7.5 означает объявление I-мерного массива X, а его длина будет определяться постепенно по мере заполнения его элементов. Специальное значение имеет переменная с именем RES - она служит для обозначения результата работы алгоритма. (Ей в качестве значения может быть присвоена также и символьная строка, например, RES ← 'NO' , что в приведенном ниже примере будет означать "система не имеет решения"). При построении выражений мы будем пользоваться обычными операциями +, -, ж, /. Еще мы будем пользоваться следующим сокращением: если, например, X ← X + Y, то вместо этого будем писать просто X+Y. Ниже будет использоваться еще одна разновидность оператора присваивания - оператор ↔ : X ↔ Y означает замену местами значений X и Y.

б) Логический оператор. Он строится с помощью операций сравнения =, ≠, >, <, ≥, ⩽ и имеет два выхода: +, если условие выполняется, и -, если условие не выполняется. Он исполь-

зуется для реализации конструкций типа IF...THEN...ELSE.

в) Оператор END. Он служит для обозначения завершения работы алгоритма на рассматриваемой ветви вычислений.

Индуктивное описание алгоритма или, точнее, описание работы алгоритма на начальном куске будет состоять из отдельных шагов, составленных из упомянутых выше операторов. Шаги мы будем нумеровать, например, I), 2), 3) ... или AI, A2), A3),... (для последнего шага вместо номера разрешается писать и многоточие, например A ...).

Упомянутое в § I индуктивное описание алгоритма нахождения наибольшего общего делителя теперь в нашем языке можно записать следующим образом (MOD (a, b) - остаток от деления a на b):

```
    INPUT: X,Y
    DESCRIPTION:
    A(I) ← X
    A(2) ← Y
I)  A(3) ← MOD(A(I), A(2))
    A(3) = 0 +──────┐
    -       │       RES ← A(2)
            │       END
    ↓       │
2)  A(4) ← MOD(A(2), A(3))
    A(4) = 0 +──────┐
    -       │       RES ← A(3)
            │       END
    ↓       │
3)  A(5) ← MOD(A(3), A(4))
    . . .
```

Дадим еще индуктивное описание алгоритма Гауса для решения систем линейных уравнений (рис. I). Уравнения мы будем задавать с помощью матрицы, например, уравнению

$$\begin{cases} 8X_1 - 3X_2 = 1 \\ 2X_1 + X_2 = 9 \end{cases}$$ соответствует матрица $$\begin{pmatrix} 8 & -3 & 1 \\ 2 & 1 & 9 \end{pmatrix}$$

Решение мы будем изображать в виде вектора; для данного примера этот вектор X = (2,5).

Очевидно, один и тот же алгоритм может иметь много индуктивных описаний. Эти описания могут отличаться длиной кусков вычислений, которые заменены многоточием.

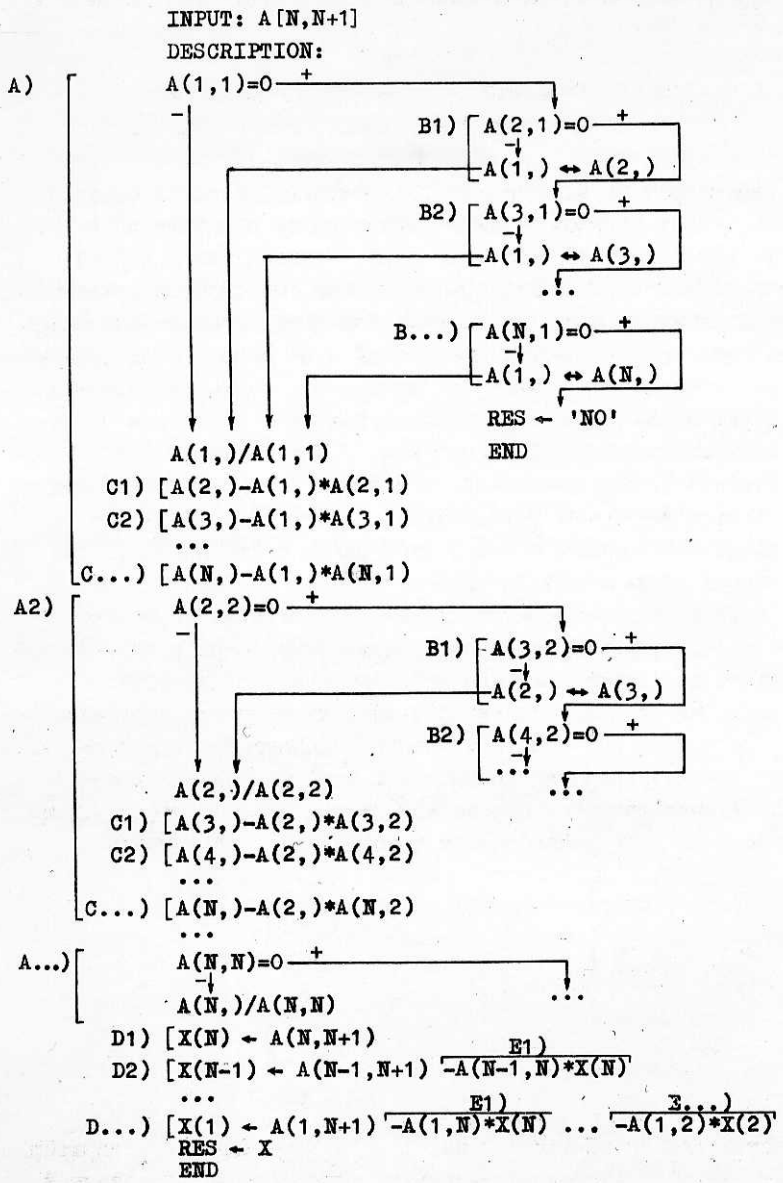

INPUT: A[N,N+1]
DESCRIPTION:
A)
A(1,1)=0
+
-
B1)
A(2,1)=0
A(1,) ↔ A(2,)
B2)
A(3,1)=0
A(1,) ↔ A(3,)
B...)
A(N,1)=0
A(1,) ↔ A(N,)
RES ← 'NO'
END
A(1,)/A(1,1)
C1) [A(2,)-A(1,)*A(2,1)
C2) [A(3,)-A(1,)*A(3,1)
C...) [A(N,)-A(1,)*A(N,1)
A2)
A(2,2)=0
B1)
A(3,2)=0
A(2,) ↔ A(3,)
B2)
A(4,2)=0
A(2,)/A(2,2)
C1) [A(3,)-A(2,)*A(3,2)
C2) [A(4,)-A(2,)*A(4,2)
C...) [A(N,)-A(2,)*A(N,2)
A...)
A(N,N)=0
A(N,)/A(N,N)
D1) [X(N) ← A(N,N+1)
D2) [X(N-1) ← A(N-1,N+1) -A(N-1,N)*X(N)
D...) [X(1) ← A(1,N+1) -A(1,N)*X(N) ... -A(1,2)*X(2)
RES ← X
END

Рис.I

Другие варианты индуктивных описаний будут рассмотрены в § 5.

§ 3. Графический DO-оператор и связанный с ним язык программирования

Синтезировать алгоритм по индуктивному описанию означает понять, каким образом заданное индуктивное описание можно обобщить для более длинных вычислений. Человек такой синтез обычно осуществляет чисто синтаксически. Он смотрит, какие части в индуктивном описании похожи, замечает простые закономерности типа арифметических прогрессий и на основе этих простейших закономерностей делает обобщения. Это означает, что человек в указанном процессе синтеза пользуется некоторым графическим языком программирования или, более точно, некоторым графическим языком обобщений, в котором семантика встречающихся объектов не играет почти никакой роли. Мы опишем один такой язык. Центральное место в этом языке будет занимать так называемый графический DO-оператор.

Графический DO-оператор служит для сокращенной записи цепочек из однотипных элементов (например, из операторов). Мы будем различать вертикальный и горизонтальный графический DO-оператор. На рис. 2 схематически изображен пример вертикального графического DO-оператора (на вертикальность оператора указывает вертикальная стрелочка в его изображении). Буква I в данном случае является переменной цикла, она меняется от 1 до 7 с шагом 2. ⊓⊔ изображает тело цикла, в нем указаны

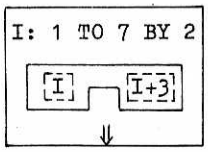


Рис. 2

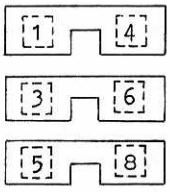


Рис. 3

те места, где встречаются I или I ± C . Указанный DO-оператор служит для обозначения вертикальной конкатенации, показанной

на рис. 3.

На рис. 4 схематически изображен пример горизонтального графического DO-оператора. Он служит для обозначения горизонтальной конкатенации, изображенной на рис. 5.

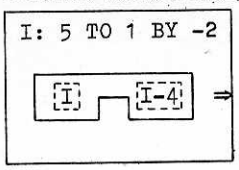


Рис. 4

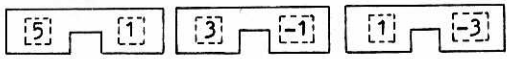


Рис. 5

В общем случае в графических DO-операторах допускаются и стрелочки, выходящие из тела цикла. Эти стрелочки будем называть горизонтальными стрелочками. Пример графического DO-оператора с горизонтальной стрелочкой схематически изображен на рис. 6.

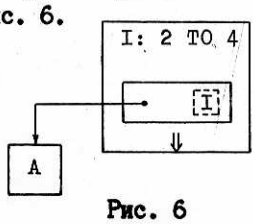


Рис. 6

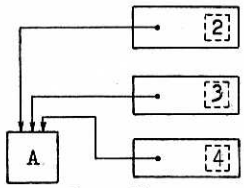


Рис. 7

Он обозначает конкатенацию, показанную на рис. 7.

Как видно из приведенных примеров, заголовок графического DO-оператора имеет вид:

переменная цикла: α TO β [BY γ]

В качестве границ цикла α и β могут быть константы, скалярные внешние переменные ± константа, переменные цикла окружающих DO-операторов ± константа, а также ∞ (если ∞ , то только в одной границе). В качестве шага цикла γ может быть только константа (если $\gamma = 1$, то его можно не писать).

На рис. 8 с помощью графического DO-оператора записана программа нахождения наибольшего общего делителя, а на рис. 9- программа решения систем линейных уравнений, соответствующая индуктивному описанию из рис. I.

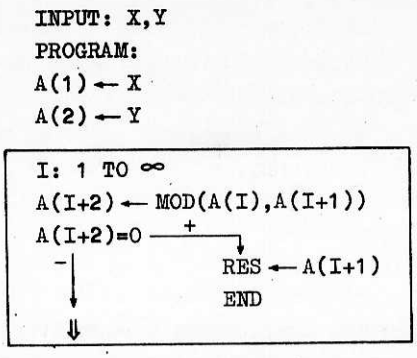

INPUT: X,Y
PROGRAM:
A(1) ← X
A(2) ← Y
I: 1 TO ∞
A(I+2) ← MOD(A(I),A(I+1))
A(I+2)=0
+
-
RES ← A(I+1)
END

Рис. 8

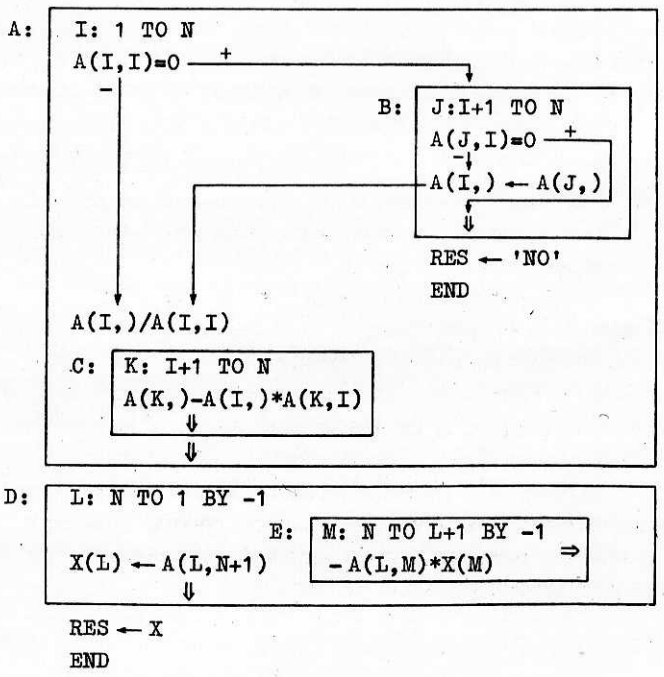

INPUT: A[N,N+1]
PROGRAM:
A:
I: 1 TO N
A(I,I)=0
+
-
B:
J:I+1 TO N
A(J,I)=0
+
-
A(I,) ← A(J,)
RES ← 'NO'
END
A(I,)/A(I,I)
C:
K: I+1 TO N
A(K,)-A(I,)*A(K,I)
D:
L: N TO 1 BY -1
E:
M: N TO L+1 BY -1
- A(L,M)*X(M)
⇒
X(L) ← A(L,N+1)
RES ← X
END

Рис. 9

§ 4. Алгоритм синтеза

Сначала дадим более точное определение индуктивного описания алгоритма. Для этого воспользуемся "обратным методом". Будем считать, что уже дана программа, которую мы хотим синтезировать, например, программа, изображенная на рис. 9. Под программой здесь имеется в виду программа, записанная с помощью графического DO-оператора. Сопоставим каждому DO-оператору этой программы некоторый идентификатор (на рис. 9 эти идентификаторы уже сопоставлены - соответственно A, B, C, D, E). Развернем теперь все DO-операторы программы в соответствии с их определениями. Эти развертки будут представлять собой конкатенацию тел DO-операторов, в которых переменные цикла заменены их значениями. Эти тела будем называть шагами индуктивного описания и будем обозначать соответственно через A1,A2,A3,..., где A - идентификатор рассматриваемого DO-оператора. В этих развертках часть шагов может быть заменена многоточием (иначе такую развертку даже невозможно изобразить, если одна из границ DO-оператора является переменной). Таким образом развертка может содержать A1,A2,... или A1,A2,...,A...) (вместо номера последнего шага разрешается писать многоточие - A...). Использование многоточия разрешается также и при изображении самих шагов - любой их фрагмент может быть заменен многоточием. Объекты, которые мы таким образом получаем из заданной программы, будем называть индуктивными описаниями алгоритма. На рис. I приведен пример индуктивного описания алгоритма, соответствующего программе, изображенной на рис. 9.

На практике мы, конечно, делаем наоборот - исходя из интуитивных соображений мы даем индуктивное описание алгоритма, а затем по этому описанию строим саму программу (так мы и делали в случае рис. I и 9). Одно индуктивное описание может отличаться от другого индуктивного описания, соответствующего той же самой программе, длиной куска развертки DO-оператора, которая заменена многоточием.

Пусть J и J' - два индуктивных описания, соответствующих одной и той же программе. Говорим, что $J \geq J'$, если в J по сравнению с J' многоточием заменены те же самые или меньшие куски разверток.

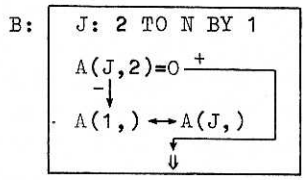
Точное описание алгоритма синтеза довольно длинное. Поэтому здесь мы ограничимся пояснением алгоритма синтеза лишь на примере.

Пусть дано индуктивное описание, изображенное на рис. I. Синтезировать программу по заданному индуктивному описанию - это фактически означает синтезировать DO-операторы по их развертками. Развертки DO-операторов, которые не содержат внутри себя развертки других DO-операторов, будем называть развертками уровня 0. Развертки, которые содержат в себе развертки уровня 0, будем называть развертками уровня I и т.д.

Синтез мы начнем с разверток уровня 0. В случае рис. I это следующие развертки:

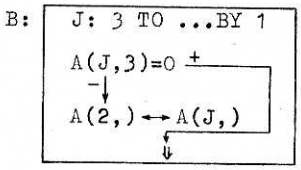
B1),B2),...,B...)	из шага AI,	(I)
B1),B2),...	из шага A2,	(2)
CI), C2),...,C...)	из шага AI,	(3)
CI), C2),...,C...)	из шага A2,	(4)
EI)	из шага D2	(5)
EI), E2),...,E...)	из шага D...	(6)

Каждой из этих разверток соответствует определенный DO-оператор. Из развертки (I) получаем DO-оператор



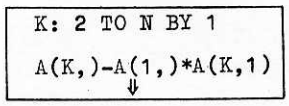
(7)

Из развертки (2), используя также развертку (I) для определения местонахождения переменной в теле цикла, получаем DO-оператор



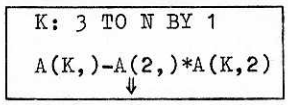
(8)

Из развертки (3) получаем DO-оператор



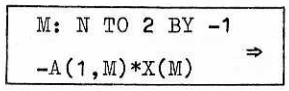
(9)

Из развертки (4) получаем DO-оператор



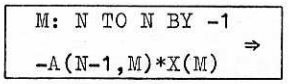
(I0)

Из развертки (6) (мы сначала берем развертку (6), так как она содержит больше членов, чем развертка (5)) получаем DO-оператор



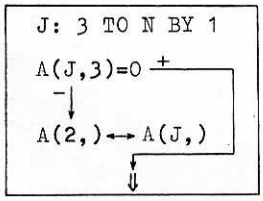
(II)

Из развертки (5), используя также развертку (6) для определения местонахождения переменной M в теле цикла, получаем DO-оператор



(I2)

Все рассмотренные выше DO-операторы построены полностью, за исключением (8'). В операторе (8') отсутствует верхняя граница. В этом случае смотрим другие DO-операторы с тем же идентификатором (в данном случае это (7)) и по ним стараемся заполнить нехватающие части. В результате мы получаем



(8)

Подставляем теперь полученные DO-операторы вместо соответствующих разверток. В результате из рис. I получаем рис. I0.

Рассмотрим теперь развертки уровня I:

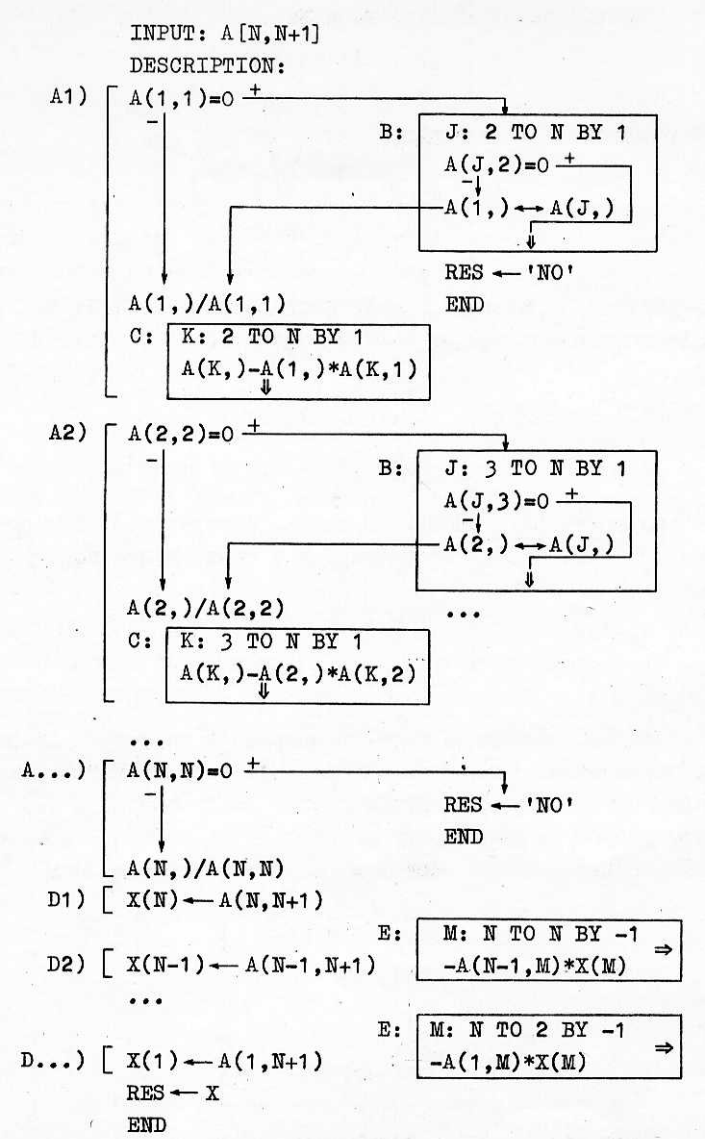

INPUT: A[N,N+1]
DESCRIPTION:
A1)
A(1,1)=0
+
-
B:
J: 2 TO N BY 1
A(J,2)=0
+
-
A(1,)↔A(J,)
RES ← 'NO'
END
A(1,)/A(1,1)
C:
K: 2 TO N BY 1
A(K,)-A(1,)*A(K,1)
A2)
A(2,2)=0
+
-
B:
J: 3 TO N BY 1
A(J,3)=0
+
-
A(2,)↔A(J,)
A(2,)/A(2,2)
...
C:
K: 3 TO N BY 1
A(K,)-A(2,)*A(K,2)
...
A...)
A(N,N)=0
+
-
RES ← 'NO'
END
A(N,)/A(N,N)
D1)
X(N) ← A(N,N+1)
E:
M: N TO N BY -1
-A(N-1,M)*X(M)
⇒
D2)
X(N-1) ← A(N-1,N+1)
...
E:
M: N TO 2 BY -1
-A(1,M)*X(M)
⇒
D...)
X(1) ← A(1,N+1)
RES ← X
END

Рис.10

AI), A2),...,A...) (I3)

D1), D2),...,D...) (I4)

Построенные выше DO-операторы мы теперь воспринимаем как отдельные операторы, входящие в эти развертки. Из шагов AI и A2 развертки (I3) мы находим тело DO-оператора A и шаг цикла. Из шагов AI и A... мы находим границы цикла. В результате мы получаем DO-оператор A, показанный на рис. 9. Аналогично из развертки (I4) мы получаем DO-оператор D . В результате мы получаем программу, изображенную на рис. 9.

Любопытно отметить, что программа, изображенная на рис.9, может быть синтезирована по значительно более короткому индуктивному описанию, показанному на рис. II, если в алгоритме синтеза использовать разумные умолчания, например, если шаг цикла не указан, то он по умолчанию равен I при условии, что нижняя граница < верхней границы, и равен -I при условии, что нижняя граница > верхней границы.

Рассмотренный выше алгоритм синтеза обозначим через Σ. Его точное определение достаточно ясно из приведенного примера. Как видно из данного примера, алгоритм Σ работает достаточно быстро. Другая не менее важная характеристика алгоритма - это его полнота. Алгоритм δ назовем полным, если для всякой программы P на рассмотренном выше языке существует такое индуктивное описание J_0, что для любого индуктивного описания $J \geq J_0$ алгоритм δ по J правильно синтезирует программу P . Имеет место следующая

ТЕОРЕМА. Алгоритм Σ является полным.

§ 5. Другие варианты индуктивных описаний

Рассмотренные выше индуктивные описания назовем сегментированными индуктивными описаниями, так как уже в самом индуктивном описании выделены шаги, которые соответствуют телу искомого DO-оператора. Для человека такое выделение шагов не вызывает больших трудностей. Тем не менее представляет интерес разработка также и методов синтеза по несегментированным индуктивным описаниям. В этом случае также удается построить достаточно эффективный эвристический алгоритм синтеза. Этот

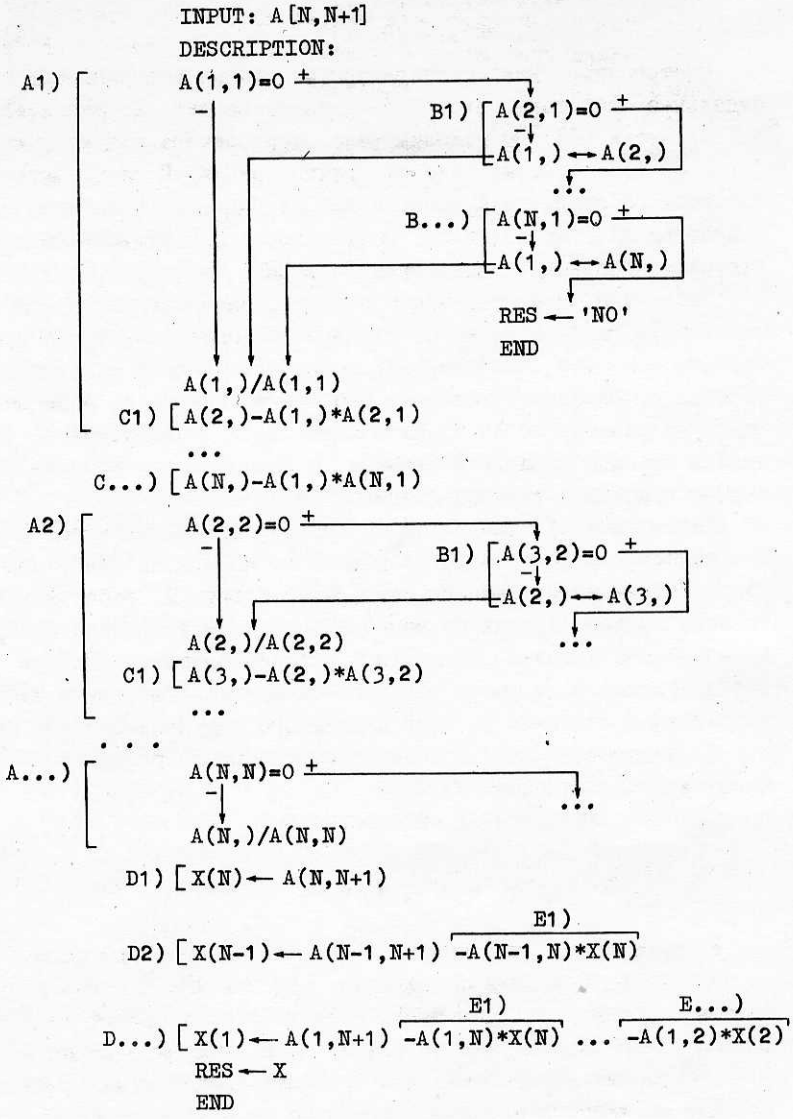

INPUT: A[N,N+1]
DESCRIPTION:
A1)
A(1,1)=0 +
-
B1)
A(2,1)=0 +
-
A(1,) ↔ A(2,)
B...)
A(N,1)=0 +
-
A(1,) ↔ A(N,)
RES ← 'NO'
END
A(1,)/A(1,1)
C1) A(2,)-A(1,)*A(2,1)
C...) A(N,)-A(1,)*A(N,1)
A2)
A(2,2)=0 +
-
B1)
A(3,2)=0 +
-
A(2,) ↔ A(3,)
A(2,)/A(2,2)
C1) A(3,)-A(2,)*A(3,2)
A...)
A(N,N)=0 +
-
A(N,)/A(N,N)
D1) X(N) ← A(N,N+1)
D2) X(N-1) ← A(N-1,N+1)
E1)
-A(N-1,N)*X(N)
D...) X(1) ← A(1,N+1)
E1)
-A(1,N)*X(N)
E...)
-A(1,2)*X(2)
RES ← X
END

Рис.II

алгоритм сначала осуществляет сегментацию индуктивного описания, а затем работает как алгоритм Σ. Фактически сегментацию и синтез он осуществляет одновременно, при том начиная с уровня 0, т.е. с самых внутренних DO-операторов. Если сегментация (разбивка на шаги) выполнена неправильно, то,как правило, по ней невозможно построить DO-оператор (например, получается, что значения варьирующихся мест в этих шагах не образуют арифметическую прогрессию, как это требуется в случае развертки DO-оператора).

Возможен еще один вариант индуктивного описания, а именно, когда индуктивное описание работы алгоритма мы даем при фиксированных значениях некоторых (или всех) входных переменных. Например, в случае алгоритма решения систем линейных уравнений вместо A[N,N+1] мы можем брать матрицу A[3, 4] и на ней демонстрировать работу алгоритма (рис. I2). Такого рода индуктивные описания мы будем называть индуктивными описаниями через примеры. Они также могут быть как сегментированными, так и не сегментированными. В случае сегментированных индуктивных описаний при изображении номеров шагов мы будем допускать записи вида A(3 = N) (см. рис. I2). Такого рода дополнительную информацию человек-составитель индуктивного описания может указать без большого труда, но она очень облегчает синтез программы. При наличии упомянутой дополнительной информации еще возможно построение эффективных эвристических алгоритмов синтеза. В случае отсутствия сегментации построение эффективных эвристических алгоритмов синтеза уже становится более трудной задачей. В этом случае она превращается в известную задачу синтеза программ по примерам, рассмотренную в работах А.Бирмана [2,3], М.Бауэра [4] и др.

§ 6. Синтез утверждений

Описанный выше индуктивный синтез может быть применен также и для синтеза утверждений о программах. Мы не будем здесь заниматься более подробными разъяснениями. Рассмотрим лишь один пример, оставляя общие выводы для читателя. На рис. I3 дано индуктивное описание алгоритма сортировки. В этом описании для начальных шагов даны утверждения $\langle \ldots \rangle$, которые

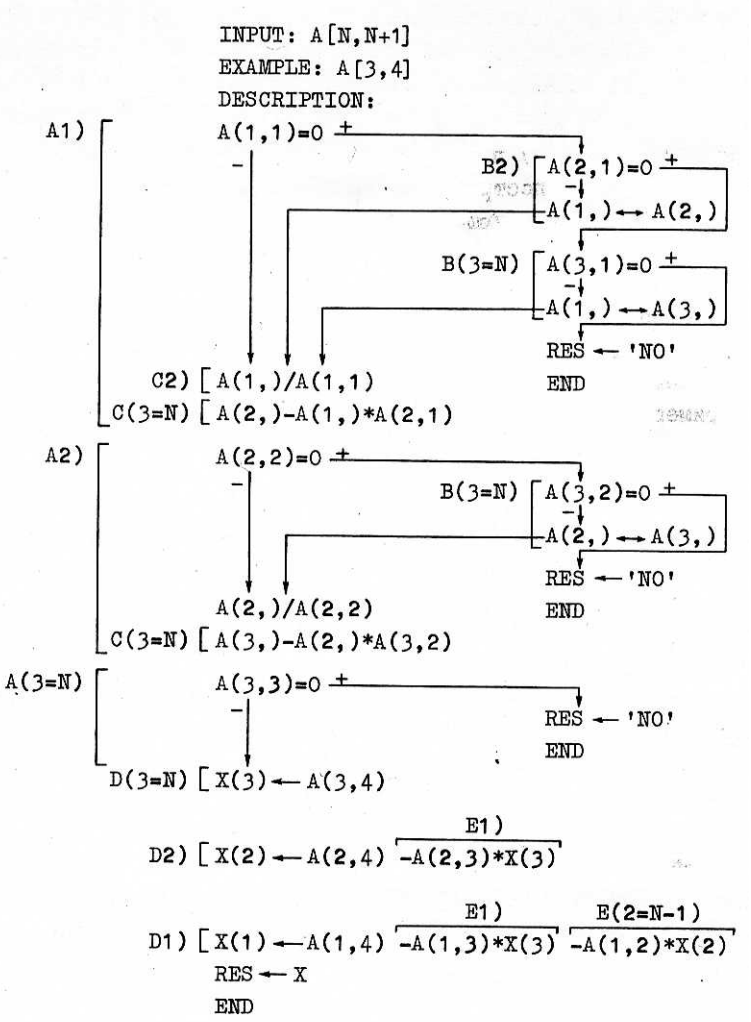

INPUT: A[N,N+1]
EXAMPLE: A[3,4]
DESCRIPTION:
A1)
A(1,1)=0
+
-
B2)
A(2,1)=0
+
-
A(1,)↔A(2,)
B(3=N)
A(3,1)=0
+
-
A(1,)↔A(3,)
RES ← 'NO'
END
C2) [A(1,)/A(1,1)
C(3=N) [A(2,)-A(1,)*A(2,1)
A2)
A(2,2)=0
+
-
B(3=N)
A(3,2)=0
+
-
A(2,)↔A(3,)
RES ← 'NO'
END
A(2,)/A(2,2)
C(3=N) [A(3,)-A(2,)*A(3,2)
A(3=N)
A(3,3)=0
+
-
RES ← 'NO'
END
D(3=N) [X(3) ← A(3,4)
E1)
D2) [X(2) ← A(2,4) -A(2,3)*X(3)
E1)
E(2=N-1)
D1) [X(1) ← A(1,4) -A(1,3)*X(3) -A(1,2)*X(2)
RES ← X
END

Рис.I2

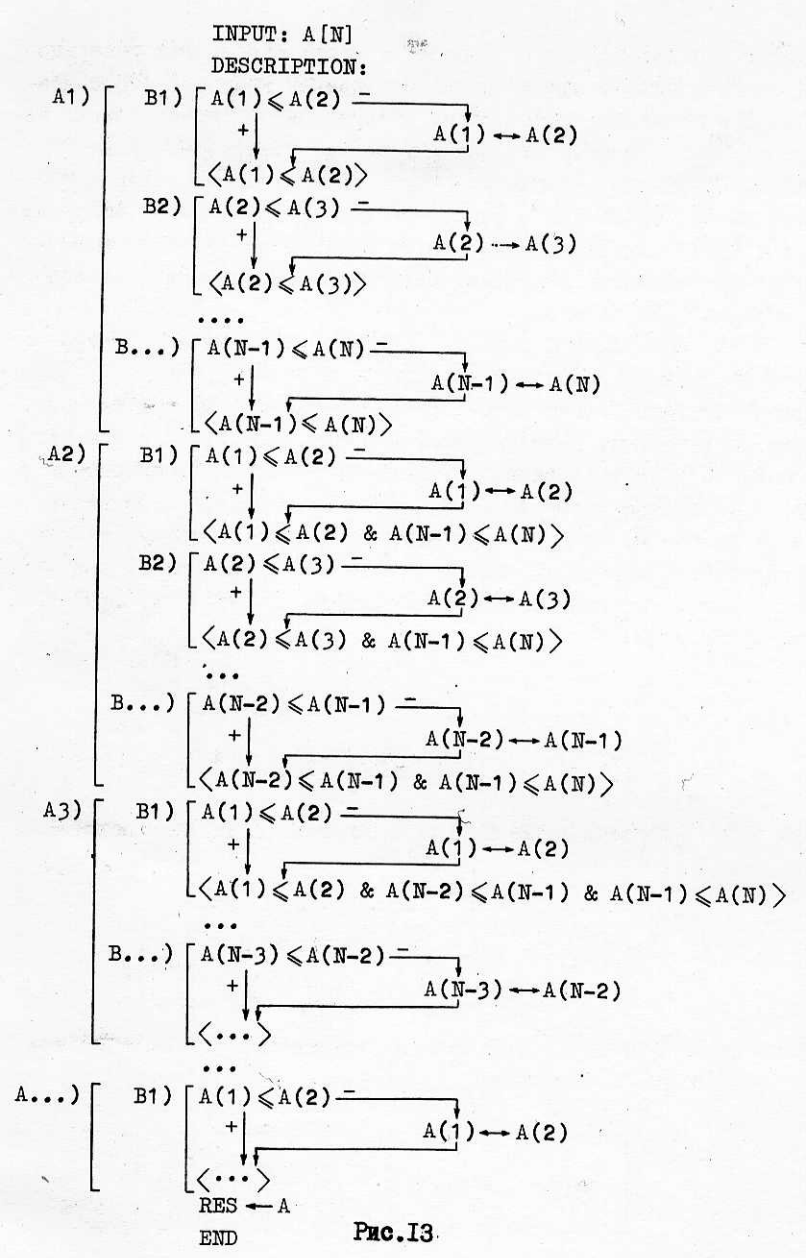

INPUT: A[N]
DESCRIPTION:
A1)
B1)
A(1)⩽A(2)
−
+
A(1)↔A(2)
⟨A(1)⩽A(2)⟩
B2)
A(2)⩽A(3)
−
+
A(2)↔A(3)
⟨A(2)⩽A(3)⟩
....
B...)
A(N-1)⩽A(N)
−
+
A(N-1)↔A(N)
⟨A(N-1)⩽A(N)⟩
A2)
B1)
A(1)⩽A(2)
−
+
A(1)↔A(2)
⟨A(1)⩽A(2) & A(N-1)⩽A(N)⟩
B2)
A(2)⩽A(3)
−
+
A(2)↔A(3)
⟨A(2)⩽A(3) & A(N-1)⩽A(N)⟩
...
B...)
A(N-2)⩽A(N-1)
−
+
A(N-2)↔A(N-1)
⟨A(N-2)⩽A(N-1) & A(N-1)⩽A(N)⟩
A3)
B1)
A(1)⩽A(2)
−
+
A(1)↔A(2)
⟨A(1)⩽A(2) & A(N-2)⩽A(N-1) & A(N-1)⩽A(N)⟩
...
B...)
A(N-3)⩽A(N-2)
−
+
A(N-3)↔A(N-2)
⟨...⟩
...
A...)
B1)
A(1)⩽A(2)
−
+
A(1)↔A(2)
⟨...⟩
RES ← A
END

Рис.I3

должны выполняться после соответствующих шагов. Эти утверждения пишет человек одновременно с заданием индуктивного описания. Для начальных шагов такие утверждения написать обычно нетрудно. Предположим, что нас интересует утверждение в конце работы программы. Рассмотрим метод синтеза этого утверждения. Этот метод, также как и описанный выше метод синтеза программ, будет чисто синтаксическим, он не будет использовать семантику рассматриваемых объектов. Поэтому он иногда может давать и неверный результат.

Итак, переходим к рис. I3. Сначала, как обычно, синтезируем внутренние DO-операторы. При этом утверждения $\langle \ldots \rangle$ воспринимаем как обычные операторы и синтезируем их вместе с другими операторами. В результате получаем рис. I4. Далее мы должны синтезировать внешний DO-оператор. Но тела DO-оператора B, соответствующие шагам A1,A2,A3, пока не являются похожими из-за того, что не являются похожими записи соответствующих утверждений. Попытаемся сделать записи этих утверждений похожими, применяя горизонтальный DO-оператор. Начнем с утверждения, входящего в A3) B:

A3)B
. . . . . .
$\langle$ A(I)$\leqslant$A(I+1) & A(N-2)$\leqslant$A(N-1) & A(N-1)$\leqslant$A(N)$\rangle$

Его можно записать с помощью DO-оператора следующим образом:

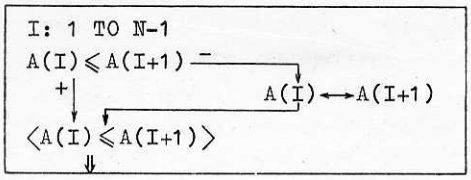
A3)B
. . . . . .
$\langle$ A(I)$\leqslant$A(I+1) K: N-2 TO N-1 & A(K)$\leqslant$A(K+1) $\Rightarrow$ $\rangle$

Применяя полученную форму записи утверждения к остальным телам оператора B, получаем:

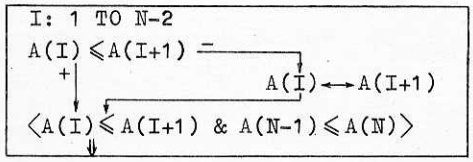
A2)B
. . . . . .
$\langle$ A(I)$\leqslant$A(I+1) K: N-1 TO N-1 & A(K)$\leqslant$A(K+1) $\Rightarrow$ $\rangle$

INPUT: A(N)
DESCRIPTION:

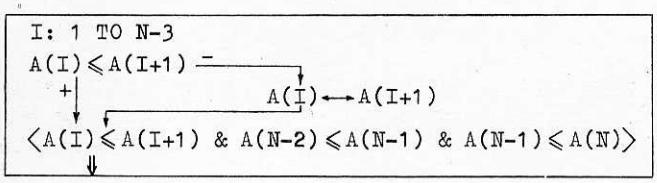
A1)B:



A2)B:

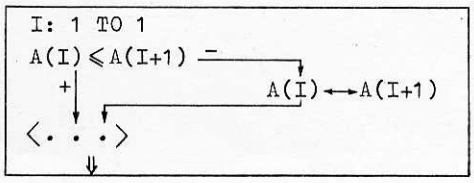


A3)B:



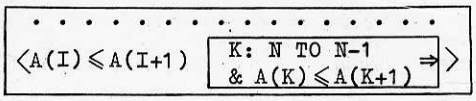
. . .

A...)B:

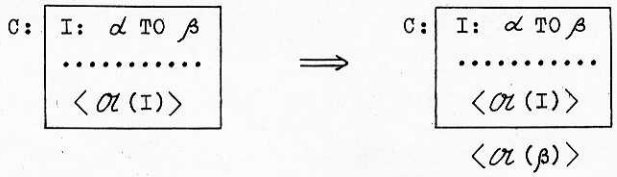


RES ← A
END

Рис.I4

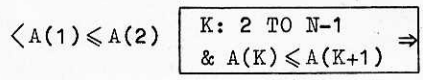


Теперь все тела оператора B являются похожими. Синтезируем теперь оператор A. В результате получаем рис. I5. Этот рисунок содержит искомую программу вместе с утверждениями. Нас интересует утверждение в конце работы программы, т.е. после оператора A. Будем пользоваться следующим правилом вывода:



т.е. если после каждого шага I оператора C стоит утверждение $\langle \alpha(I) \rangle$, то после всего оператора разрешается добавить утверждение $\langle \alpha(\beta) \rangle$. В результате из рис. I5 получаем рис. I6.

Таким образом, после оператора A мы получаем утверждение



т.е.

$$\langle A(1) \leqslant A(2) \& A(2) \leqslant A(3) \& \ldots \& A(N-1) \leqslant A(N) \rangle .$$

В данном случае полученное утверждение очевидно является верным.

Ясно, что описанный выше способ вывода можно также формализировать. Хотя и этот способ вывода не является абсолютным, но на реальных примерах он обычно дает правильный результат. Наверно, человек в процессе интуитивной верификации программ тоже пользуется чем-то подобным.

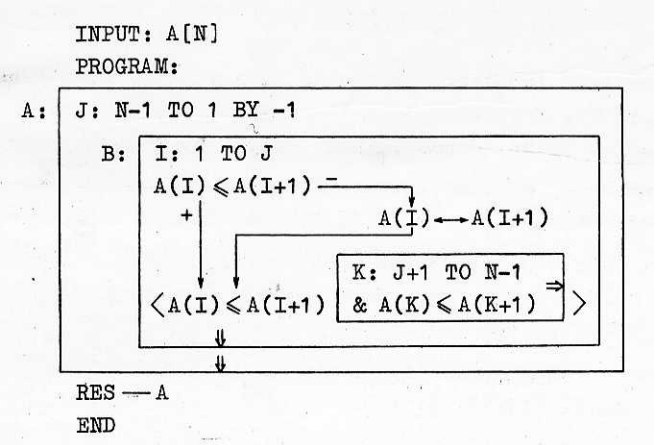

INPUT: A[N]
PROGRAM:
A:
J: N-1 TO 1 BY -1
B:
I: 1 TO J
A(I)⩽A(I+1)
–
+
A(I)←→A(I+1)
K: J+1 TO N-1
& A(K)⩽A(K+1)
⇒
⟨A(I)⩽A(I+1)⟩
RES — A
END

Рис.I5

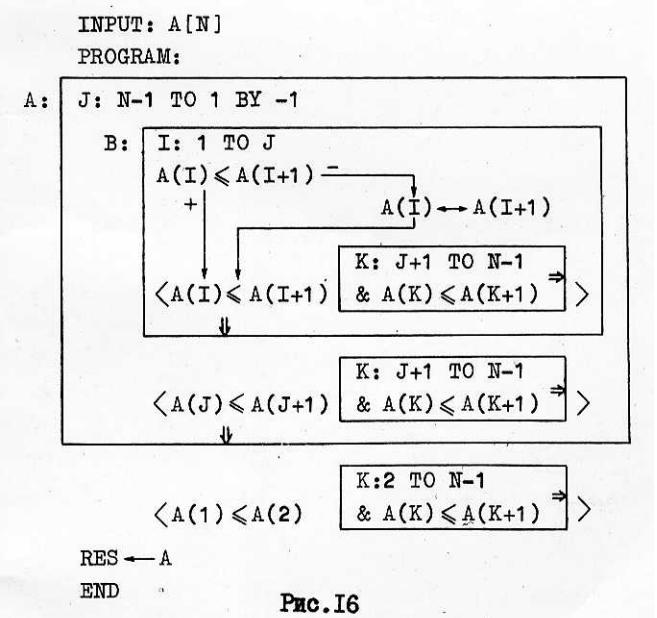

INPUT: A[N]
PROGRAM:
A:
J: N-1 TO 1 BY -1
B:
I: 1 TO J
A(I)⩽A(I+1)
–
+
A(I)←→A(I+1)
K: J+1 TO N-1
& A(K)⩽A(K+1)
⇒
⟨A(I)⩽A(I+1)⟩
K: J+1 TO N-1
& A(K)⩽A(K+1)
⇒
⟨A(J)⩽A(J+1)⟩
K:2 TO N-1
& A(K)⩽A(K+1)
⇒
⟨A(1)⩽A(2)⟩
RES ← A
END

Рис.I6

ЛИТЕРАТУРА

1. J.M.Barzdin. Inductive inference of automata, functions and programs. Proceedings of the International Congress of Mathematicians, Canada, 1974, vol.2, p. 455-460.
2. A.W.Biermann. Approaches to automatic programming. -In: Advances in Computers, vol.15, N.Y. Academic, 1976, p.1-6
3. A.W.Biermann. Constructing programs from example computa tions. IEEE Trans. of Software Engineering, vol. SE-2, no.3 (1976), p. 141-153.
4. M.A.Bauer. Programming by examples. Artificial Intelligence, vol.12 (1979), p. 1-21.