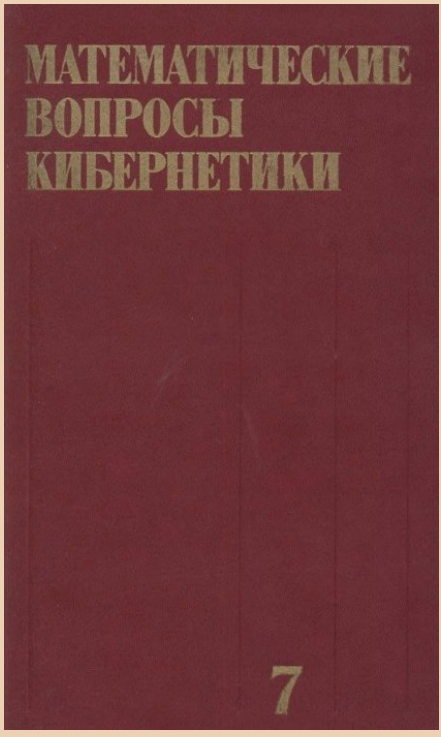
ИПМ им. М.В. Келдыша РАН • Электронная библиотека
Математические вопросы кибернетики • Выпуск 7



Р. И. Подловченко

От схем Янова к теории моделей программ

Рекомендуемая форма библиографической ссылки: Подловченко Р. И. От схем Янова к теории моделей программ // Математические вопросы кибернетики. Вып. 7. — М.: Физматлит, 1998. — С. 281–302. URL: http://library.keldysh.ru/mvk.asp?id=1998-281

ОТ СХЕМ ЯНОВА К ТЕОРИИ МОДЕЛЕЙ ПРОГРАММ *)

Р. И. ПОДЛОВЧЕНКО

(МОСКВА)

Статья посвящена 40-летию опубликования работ А. А. Ляпунова [6] и Ю. И. Янова [23]. Ими было открыто новое научное направление — теория схем программ, ознаменовавшее зарождение теоретического программирования. Проблематика этого направления складывалась в русле фундаментальных работ А. А. Ляпунова и С. В. Яблонского по теоретической кибернетике, проводимых в то же время, но опубликованных несколько позже [7, 22]. К настоящему времени среди различных областей теоретического программирования имеется раздел, непосредственно развивающий работы [6, 23]. Это — теория моделей программ. Цель данной статьи — продемонстрировать связь этой теории с работами [6, 23], осветить ее проблематику, методику решения основных проблем, отметить связь моделей программ с другими моделями вычислений. Выборочно, для обоснования направлений, в которых развивается теория, приводятся результаты по решению ее задач. Мы отталкиваемся от понятий и проблематики статей [6, 23], излагая их неформально и с позиций, созвучных нашему времени.

§ 1. Схемы Янова и полученные для них результаты

Введенные в [6, 23] объекты исследований вошли в теорию под названием схем Янова. В современном описании *схема Янова* представляет собой конечный граф специальной структуры, размеченный над двумя конечными непустыми алфавитами: Y и P. Элементы первого называются *операторными символами*, второго — *логическими переменными*; такая переменная принимает два значения: 0 и 1. Вершины графа, задающего схему, подразделяются на *вход* (с одной исходящей дугой и без входящих дуг), *выход* (без исходящих из него дуг), *преобразователи*, каждый из которых имеет одну исходящую дугу и помечен символом из Y, и *распознаватели*, каждый из которых помечен переменной из P и имеет две исходящие дуги с метками 0 и 1 соответственно.

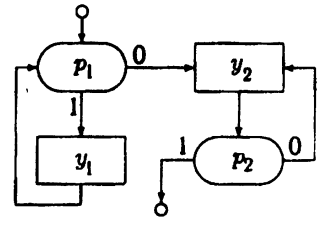


Рис. 1

Пример схемы дан на рис. 1. Она структурно описывает программу с двумя операторами, обозначенными y_1 и y_2, и двумя булевыми выражениями, обозначенными p_1 и p_2.

*) Работа выполнена при поддержке Российского фонда фундаментальных исследований (проект 97-01-00975).

Функциональное описание схемы осуществляется введением процедуры ее выполнения. Такая процедура призвана прокладывать в схеме путь, начинающийся во входе с пустой цепочкой операторных символов и сопровождающийся приписыванием к текущей цепочке операторного символа, сопоставленного пройденному преобразователю. Пути в схеме ветвятся при прохождении через распознаватель, поэтому для однозначности выбора дальнейшего направления пути нужно знать значение сопоставленной этому распознавателю переменной на текущей операторной цепочке — тогда выбирается дуга, помеченная этим значением. Процедура выполнения схемы должна быть применимой к любой схеме, поэтому вторым ее аргументом является функция, которая каждой цепочке из Y^* приписывает значения всех логических переменных. Она называется *функцией разметки*. Множество всех функций разметки обозначается $\mathscr{L}$.

Говорим, что *схема G останавливается на функции μ из $\mathscr{L}$*, если ее выполнение на μ привело в выход схемы. Накопившуюся к этому моменту цепочку называем *результатом выполнения G на μ* и обозначаем $G(\mu)$. Альтернативный случай: схема G «зацикливается» и значение $G(\mu)$ не определено.

В [23] рассматриваются специальным образом построенные подмножества L множества $\mathscr{L}$ и для каждого из них вводится *L-эквивалентность* схем G_1 и G_2:

$$(G_1 \overset{L}{\sim} G_2) \Longleftrightarrow (\forall \mu \in L)\ (G_1(\mu) = G_2(\mu)).$$

Отдельное подмножество L индуцируется заданием функции, приписывающей всякому операторному символу из Y свое подмножество логических переменных из P; оно называется *сдвигом символа*, а сама функция — *распределением сдвигов*. Пусть s — заданное распределение сдвигов; тогда множество L_s по определению состоит из всех функций разметки μ, которые удовлетворяют такому требованию: каковы бы ни были операторная цепочка h и символ y из Y, значения μ на h и на hy совпадают для всех переменных, не принадлежащих сдвигу sy.

Способ задания множества L_s основан на следующем соображении. Если y обозначает оператор присваивания с левой частью r, то после выполнения оператора y могут измениться значения только тех булевых выражений, которые используют переменную r; соответствующие этим булевым выражениям логические переменные и составляют сдвиг символа y.

Если сдвиг любого символа из Y состоит из всего множества P, множество L_s совпадает с $\mathscr{L}$. В этом случае эквивалентность схем называется *сильной*.

В качестве основной в [6, 23] выдвинута проблема построения системы *эквивалентных преобразований* схем, полной в некотором классе схем. Содержательно под полнотой системы преобразований в классе $\mathscr{M}$ понимается следующее ее свойство: для любых двух эквивалентных схем из $\mathscr{M}$ существует цепочка преобразований, принадлежащих системе и переводящих одну схему в другую. Для проверки полноты предъявляемой системы строится алгоритм, который получает на вход две произвольные схемы из $\mathscr{M}$, в случае их эквивалентности преобразует схемы в изоморфные друг другу, а при неэквивалентности просто останавливается, убедившись в этом. Позднее (см. [9]) такой алгоритм был назван *алгоритмом канонизации пар схем из $\mathscr{M}$*.

Таким образом, проблема построения полной системы фактически рассматривается в том случае, когда в $\mathscr{M}$ разрешима проблема эквивалентности, т. е. если существует алгоритм, распознающий, эквивалентны ли поступившие на его вход две произвольные схемы из $\mathscr{M}$. Этот подход закрепился и в дальнейшем, в силу чего проблема эквивалентности, имеющая и самостоятельное значение, приобрела дополнительный интерес.

В [23] проанализирован класс схем, каждая из которых использует любой операторный символ не более одного раза. Обозначим этот класс через $\mathcal{M}_0$. Основной результат для него дается теоремой 1.

Теорема 1. *Для любого распределения сдвигов s существует система L_s-эквивалентных преобразований схем, полная в классе $\mathcal{M}_0$.*

Но полнота системы устанавливается алгоритмом канонизации пар схем, поэтому из теоремы 1 следует разрешимость проблемы L_s-эквивалентности в $\mathcal{M}_0$.

Заметим, что исследованиями, предпринятыми после опубликования статьи [23], было показано, что утверждение теоремы 1 сохраняет свою силу при замене $\mathcal{M}_0$ на множество всех схем над Y и P.

§ 2. Модели рекурсивных программ

Перейдем к обобщению схем Янова, сохраняя применяемый в [23] подход к описанию их функционирования. Отметим, что схемы Янова являют собой схемы таких программ, которые используют все принятые средства композиции операторов, кроме одного: оператора процедуры. Так, например, в схеме, показанной на рис. 1, представлено произведение цикла с предусловием p_1 на цикл с постусловием p_2. Напрашивается введение оператора процедуры. Для этого граф, описывающий ранее всю программу, а теперь — главную или ведущую программу, пополняется подграфами, описывающими процедуры. Но процедуры могут быть рекурсивными, а потому мы получаем в общем случае схему рекурсивной программы.

Далее, при определении эквивалентности схем наряду с варьированием множества L можно варьировать отношение эквивалентности операторных цепочек, заменяя используемое в [23] их равенство некоторой эквивалентностью ν. Ее введением отражается то обстоятельство, что различные последовательности операторов, будучи применены к одному и тому же набору данных, могут привести к одинаковым результатам.

Опишем теперь строго эти обобщения.

Схема программы с процедурами (далее — схема рекурсивной программы или просто схема) строится над конечным базисом, состоящим из элементов четырех непустых и непересекающихся алфавитов: Y, C, R и P. Элементы первых трех называются *символами*, элементы множества P — *логическими переменными*; каждая логическая переменная принимает значения 0 и 1. Символами обозначаются операторы, логическими переменными — булевы выражения.

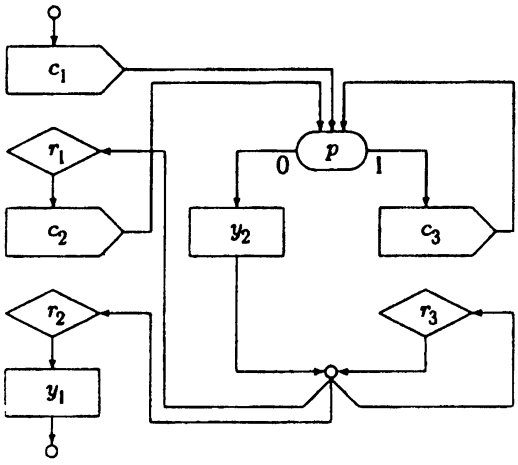
Схема представляет собой размеченный над базисом конечный ориентированный граф следующего строения. Граф распадается на подграфы с непересекающимися множествами вершин, в сумме составляющими все вершины графа. Подграфы соединены между собой дугами. Опишем строение подграфов и способ их соединения.

Один из подграфов называется *главным*; в нем выделены *вход* (вершина без входящих дуг и с одной исходящей) и *выход* (вершина без исходящих из нее дуг). В любом неглавном подграфе тоже выделены две вершины: *инициальная* и *финальная*. Всякая вершина (кроме входа, выхода и финальных вершин) относится к одному из четырех типов: *преобразователь*, *распознаватель*, *вызов*, *возврат*. Из распознавателя исходят две дуги, помеченные числами 0 и 1 соответственно; из преобразователя, вызова, возврата — по одной дуге. Распознавателю сопоставлена переменная из P, преобразователю, вызову, возврату — символ из Y, C, R соответственно. Вызовы и возвраты находятся во взаимно однозначном соответствии. Вызов и соответствующий ему возврат составляют пару, принадлежащую одному и тому же подграфу. Всякой такой паре присвоен номер, отличный

от номеров других пар. Дуга из вызова ведет в инициальную вершину своего или чужого подграфа, и тогда из финальной вершины этого подграфа исходит дуга, ведущая в соответствующий вызову возврат, причем это — единственная приходящая в него дуга. Иных дуг, кроме упомянутых, из финальной вершины не исходит. Дуги, исходящие из вершин иного типа, ведут в вершины того же подграфа, которому принадлежит их начало.

На рис. 2 представлена схема, содержащая два подграфа. Главный подграф находится слева и содержит, кроме входа и выхода, две пары вершин вызов/возврат, помеченные символами c_1, r_1 и c_2, r_2 соответственно, и один преобразователь, помеченный символом y_1. Этот подграф соединен с другим подграфом двумя дугами, исходящими из вызовов, и двумя дугами, приходящими в возвраты. Схеме принадлежат три пары соответствующих друг другу вызовов и возвратов: две в главном подграфе, одна — в неглавном.

Легко заметить, что схема Янова является частным случаем схемы рекурсивной программы — последняя исчерпывается главным подграфом. Далее она называется *схемой простой программы.*



Обозначения:

○ — входы, выходы и финальные вершины;

— преобразователь; — вызов;

— распознаватель; — возврат.

Сопоставленные вершинам элементы базиса вписаны в фигуры, изображающие вершины.

Рис. 2

Функционирование схемы осуществляется на функциях разметки. Введем понятие последней.

Слово в алфавите $Y \cup C \cup R$ называется *цепочкой*; цепочка считается *правильной*, если в любом ее префиксе число вхождений символов из R не превосходит числа вхождений символов из C. Обозначим через H множество всех правильных цепочек. Пусть

$$X = \{x \mid x\colon P \to \{0, 1\}\}.$$

Функцией разметки назовем отображение $\mu\colon H \to X$. Множество всех функций разметки обозначим через $\mathfrak{L}$.

Пусть $\mu \in \mathfrak{L}$. *Выполнением схемы на функции* μ назовем процесс, состоящий в путешествии по схеме и сопровождающийся построением правильной цепочки. Для однозначности выбора пути, кроме μ, используется магазин, в который загружаются номера вызовов в схеме. Путешествие начинается по дуге, исходящей из входа, при пустых магазине и цепочке. Переход через вершину с сопоставленным ей символом сопровождается приписыванием этого символа к текущей цепочке справа. Если переходимая вершина — вызов, то в магазин загружается его номер. При переходе через финальную вершину и распознаватель текущая цепочка не возрастает. В первом случае из макушки магазина, который заведомо непуст, извлекается номер, и путешествие продолжается по дуге, ведущей к возврату с этим номером. При переходе через распознаватель используется функция μ: в качестве следующей берется дуга, несущая метку $\mu h(p)$, где p — сопоставленная распознавателю переменная, а h — текущая цепочка. При достижении выхода путешествие прекращается. В этом случае говорим, что *схема остановилась на* μ и построенную цепочку называем *результатом* ее выполнения на μ. Путь, пройденный в схеме при ее выполнении на μ, именуем *прокладываемым в схеме функцией* μ.

Пусть ν — эквивалентность в множестве H, и пусть $L \subseteq \mathscr{L}$. Схемы G_1 и G_2 назовем (ν, L)-*эквивалентными*, если, какова бы ни была функция из L, всякий раз, как на ней останавливается одна из схем, останавливается и другая, и в этом случае результатами их выполнения являются ν-эквивалентные цепочки.

Множество всех схем над выбранным базисом, рассматриваемое вместе с (ν, L)-эквивалентностью схем, называется (ν, L)-*моделью*.

Если ν — это отношение тождества цепочек, а L совпадает с $\mathscr{L}$, то (ν, L)-модель называется *максимальной* в силу того, что из эквивалентности схем в этой модели следует их эквивалентность в любой другой.

Множество всех схем простых программ, рассматриваемое вместе с (ν, L)-эквивалентностью, именуется *простой подмоделью* (ν, L)-*модели*. Нетрудно заметить, что простая подмодель максимальной модели представляет собой множество схем Янова с сильной эквивалентностью в нем. Из сильной эквивалентности схем простых программ следует их эквивалентность в любой простой подмодели, поэтому простая подмодель максимальной модели тоже называется *максимальной*.

Простая модель может быть определена и независимо от модели рекурсивных программ. Для этого нужно рассматривать только схемы простых программ и при введении эквивалентности схем в качестве параметров брать эквивалентность в множестве Y^* и так называемые *простые* функции разметки (такая функция описывалась в § 1, она отображает Y^* в X). Далее множество всех простых функций разметки будем обозначать $\mathscr{L}^Y$.

По любой простой модели можно построить модель рекурсивных программ, подмоделью которой является первая. И поэтому все простые модели, исследованные в [23], представляют собой частные случаи подмоделей, введенных в § 2.

Говоря далее о моделях, мы будем иметь в виду модели рекурсивных программ. Те случаи, когда модель проста, всегда будут отмечаться специально. Параметры простой модели обозначаем по-прежнему: ν и L.

§ 3. Отбор моделей рекурсивных программ

Отбор моделей рекурсивных программ подчиняется следующим двум задачам.

Задача 1. Выделить модели, пригодные для разработки в них эквивалентных преобразований программ; при этом под пригодностью понимается то свойство, что эквивалентное преобразование схемы одновременно является эквивалентным преобразованием программы, описываемой этой схемой.

Задача 2. Отобрать модели (обладающие полными системами эквивалентных преобразований схем), для которых справедливо следующее утверждение. Для любых двух моделей из числа отобранных можно, не предъявляя самих систем, а лишь сравнивая параметры моделей, судить о том, сравнимы ли системы по богатству представленных в них преобразований.

Строгая постановка первой задачи требует определения программы, о которой идет речь, и введения отношения «класс программ аппроксимируется моделью» («модель аппроксимирует класс программ»).

При определении программы исходим из следующего положения: структурно программа и соответствующая ей схема совпадают. Этим обеспечивается важное свойство, состоящее в том, что всякое преобразование схемы одновременно является и преобразованием программы, описываемой этой схемой.

Таким образом, структура программы определена.

Функциональная трактовка программы основана на *семантике базиса*. Так называется алгебраическая система σ, использующая произвольно выбранное множество Σ и сопоставляющая каждому элементу b базиса всю-

ду определенное отображение σb, тип которого определяется следующим образом:

$$\begin{cases} \sigma b\colon \Sigma \to \Sigma, & \text{если } b \in Y \cup C, \\ \sigma b\colon \Sigma \times \Sigma \to \Sigma, & \text{если } b \in R, \\ \sigma b\colon \Sigma \to \{0, 1\}, & \text{если } b \in P. \end{cases}$$

Элементы множества Σ называются *состояниями памяти.*

При заданной семантике σ определяется *процедура выполнения программы на состоянии* ξ_0, где $\xi_0 \in \Sigma$. Она, как и в случае выполнения схемы на функции разметки, состоит в путешествии по программе. Последнее начинается во входе при состоянии памяти ξ_0. Чтобы путешествие было детерминированным, вводится магазин, в котором запоминаются пары вида (номер вызова, состояние памяти). Перед началом путешествия магазин пуст. Цель путешествия — в переработке текущего состояния памяти. Она осуществляется при переходе через вершины типа «преобразователь», «вызов», «возврат» в соответствии с тем, как в семантике σ трактуются символы b, сопоставленные этим вершинам. Если b принадлежит $Y \cup C$, то преобразование текущего состояния не требует пояснений. Отметим только, что при переходе через вызов магазин загружается парой (номер вызова; состояние памяти, предшествующее его прохождению). При переходе через финальную вершину с макушки магазина берется хранящаяся там пара, по номеру вызова определяется подлежащий выполнению возврат, а состояние памяти, записанное в снятой из магазина паре, вкупе с текущим состоянием позволяет применить преобразование, приписанное символу b в этом случае. Переход через распознаватель с символом p сопровождается вычислением значения σp на текущем состоянии памяти и выбором дуги, помеченной вычисленным значением, как направления путешествия. Путешествие завершается, если оно приводит в выход программы; тогда говорим, что *программа остановилась на паре* (σ, ξ_0), а состояние памяти, при котором это произошло, называем *заключительным.*

Таким образом, при заданной семантике σ программа реализует частичное отображение множества Σ в себя; это отображение определено для ξ_0 из Σ в том и только том случае, когда программа останавливается на паре (σ, ξ_0), и сопоставляет тогда состоянию ξ_0 заключительное состояние.

Две программы назовем *σ-эквивалентными* в том и только том случае, если они осуществляют одно и то же отображение множества Σ в себя. Множество программ с отношением их σ-эквивалентности назовем *σ-классом* программ.

Подчеркнем следующее. Определенные нами программы можно квалифицировать как абстрактные. Но если используемые реальной программой переменные рассматривать как память, а выполнение программы воспринимать как преобразование состояний памяти в себя, то мы придем к некоторой программе из числа определенных нами. Таким образом, при изложенном взгляде на реальную программу можно утверждать, что их множество «утопает» в σ-классах программ. Подробнее о связи реальных программ с абстрактными говорится в [16].

По определению, (ν, L)-модель *аппроксимирует* σ-класс программ, если из эквивалентности схем в модели следует σ-эквивалентность соответствующих им программ. Если данное отношение выполнено, то обеспечено следующее: всякое (ν, L)-эквивалентное преобразование схемы является и σ-эквивалентным преобразованием соответствующей ей программы. Таким образом, пригодными для разработки эквивалентных преобразований программ следует признать аппроксимирующие модели, т. е. такие, для которых существуют аппроксимируемые ими классы программ.

В [12] при решении задачи 1 используется понятие строгой аппроксимации. Говорим, что (ν, L)-модель является *строго аппроксимирующей*, если

существует такое непустое множество семантик S базиса, что (ν, L)-эквивалентность схем влечет σ-эквивалентность соответствующих программ для любой семантики σ из S и, обратно, σ-эквивалентность программ при любой σ из S влечет (ν, L)-эквивалентность соответствующих им схем.

В [12] наложением определенных требований на параметры ν и L введено понятие полугрупповой модели. Там же доказана

Теорема 2. *Всякая полугрупповая модель рекурсивных программ — строго аппроксимирующая.*

Заметим, что в [16] приводится более подробное, чем в [12], доказательство этой теоремы.

Ниже нам потребуются простые подмодели полугрупповых моделей. Они являются строго аппроксимирующими для простых программ, что устанавливается в [8], и тоже называются полугрупповыми. Дадим определение простой полугрупповой модели.

Простая (ν, L)-модель называется *полугрупповой*, если ν является полугрупповой эквивалентностью, а множество L состоит из ν-согласованных функций разметки и замкнуто по сдвигу. Эквивалентность ν по определению является *полугрупповой*, если для любых цепочек h_1, h_2, h_3, h_4 из Y^* выполнено соотношение

$$(h_1 \overset{\nu}{\sim} h_2) \,\&\, (h_3 \overset{\nu}{\sim} h_4) \Longrightarrow (h_1 h_3 \overset{\nu}{\sim} h_2 h_4).$$

Функция разметки μ называется *ν-согласованной*, если для любых h_1 и h_2 из Y^* имеет место импликация

$$(h_1 \overset{\nu}{\sim} h_2) \Longrightarrow (\mu h_1 = \mu h_2).$$

Операция *сдвига функции μ на цепочку h*, где $h \in Y^*$, дает функцию разметки, обозначаемую μ_h и определяемую следующим образом: для любой цепочки g из Y^* положим $\mu_h g = \mu hg$.

В случае, когда ν — полугрупповая эквивалентность, справедливо утверждение: фактормножество полугруппы Y^* по отношению ν само является полугруппой. В связи с этим наряду с термином «полугрупповая эквивалентность ν» используется термин «отношение ν определяет в Y^* полугруппу».

Перейдем теперь к рассмотрению задачи 2. Начнем с уточнения некоторых понятий.

Пусть в множестве схем фиксировано какое-либо отношение эквивалентности. Упорядоченную пару (G_1, G_2), где G_1 и G_2 — схемы, назовем *преобразованием схемы G_1 в схему G_2*. Если схемы G_1 и G_2 эквивалентны, преобразование (G_1, G_2) назовем *эквивалентным*. Всякую совокупность T, состоящую из преобразований схем, именуем системой преобразований. Системе T сопоставим ее замыкание $\widetilde{T}$, состоящее из всех таких и только таких преобразований (G', G''), для которых существует последовательность преобразований

$$(G_1, G_2), (G_2, G_3), \ldots, (G_{n-1}, G_n), \qquad n \geqslant 2,$$

принадлежащих системе T, причем $G' = G_1$ и $G'' = G_n$. Говорим, что *система T_1 не богаче системы T_2*, если $\widetilde{T}_1 \subseteq \widetilde{T}_2$. По определению, система T, состоящая из эквивалентных преобразований схем, *полна в классе $\mathcal{M}$ схем*, если $\widetilde{T}$ совпадает с множеством всех пар эквивалентных схем, принадлежащих $\mathcal{M}^2$.

Заметим, что в этих определениях схемы можно заменить программами.

Говорим, что *одна модель аппроксимирует другую*, если из эквивалентности схем в первой модели следует их эквивалентность во второй. Например, максимальная модель аппроксимирует любую другую.

Смысл отношения аппроксимации выявлен в следующем утверждении.

Утверждение 1. *Пусть K_i — модель, ε_i — отношение эквивалентности схем в K_i, а T_i — полная в $\mathcal{M}$ система ε_i-эквивалентных преобразований схем, $i = 1, 2$. Тогда если K_1 аппроксимирует K_2, то T_1 не богаче, чем T_2.*

Если модели K_1 и K_2 аппроксимируют один и тот же класс программ, этим утверждением можно пользоваться, чтобы из двух систем выбирать более богатую. Так возникает задача, состоящая в отборе моделей, для которых отношение аппроксимации одной моделью другой модели выражается в виде требований к их параметрам.

Условимся модель с параметрами ν и L обозначать $K(\nu, L)$. Решение задачи 2 дано в [12]; его выражает

Теорема 3. *Если $K(\nu_1, L_1)$ и $K(\nu_2, L_2)$ — гладкие модели, то первая аппроксимирует вторую в том и только том случае, когда*

$$(\nu_1 \to \nu_2) \,\&\, (L_1 \supseteq L_2).$$

Гладкой здесь называется модель $K(\nu, L)$ с гладким L. Определим понятие гладкости L.

Пусть h — операторная цепочка из H. Тогда *h-графиком* назовем отображение, сопоставляющее каждому префиксу слова h, включая пустое слово и само h, некоторый (вообще говоря, свой) элемент из X. Говорим, что *h-график принадлежит функции* μ, где $\mu \in \mathcal{L}$, если на любом префиксе цепочки h значение h-графика совпадает со значением функции μ. Обозначим через $Z(\mu)$ множество всех h-графиков, принадлежащих μ.

Для произвольного множества Z, состоящего из h-графиков, используем обозначение $l(Z) = \{\mu \mid Z(\mu) \subseteq Z\}$.

Множество L называется *гладким*, если при $Z = \bigcup_{\mu \in L} Z(\mu)$ выполнено равенство $l(Z) = L$.

Содержательно гладкое множество можно описать следующим образом. Если функции множества L «рассыпать» на h-графики, а потом из полученного множества h-графиков «конструировать» функции разметки, то полученное множество функций совпадет с L.

В заключение § 3 сформулируем теорему 4.

Теорема 4. *Множество гладких моделей с отношением аппроксимации является верхней полурешеткой, обладающей максимальным элементом; им является максимальная модель.*

Из этой теоремы следует

Утверждение 2. *Простые гладкие модели составляют верхнюю полурешетку по отношению аппроксимации. Схемы Янова с сильной эквивалентностью являются максимальным элементом этой полурешетки.*

§ 4. Задание параметров модели

Сформулируем необходимые требования к параметрам модели $K(\nu, L)$. Сначала обсудим эквивалентность ν. Нас интересуют модели с разрешимой проблемой эквивалентности, поэтому следует брать только такие ν, которые сами являются *разрешимыми в H*, т. е. для которых существует алгоритм, устанавливающий для любых двух цепочек из H, являются ли они ν-эквивалентными.

Теперь обратимся к выбору множества L при заданном ν. Обзор исследованных к настоящему времени моделей показывает, что при этом выборе действуют три излагаемых ниже положения.

Положение 1. Рассматриваются только полугрупповые модели, ибо они гарантированно аппроксимируют программы.

В полугрупповой модели $K(\nu, L)$, согласно ее определению, множество L состоит из ν-согласованных функций разметки и является замкнутым по сдвигу. Понятия ν-согласованности функции разметки и сдвига функции разметки на цепочку получаются из данных выше (для простых функций разметки) заменой множества Y^* множеством H. Обозначим через $\mathcal{L}_\nu$ множество всех ν-согласованных функций разметки. Таким образом, в полугрупповой модели имеем

$$L = \mathcal{L}_\nu \cap L',$$

где $L' \subseteq \mathcal{L}$. Множество $\mathcal{L}_\nu$ замкнуто по сдвигу, поэтому для замкнутости по сдвигу множества L достаточно потребовать, чтобы этим же свойством обладало L'. Итак, положение 1 сводит выбор множества L к выбору замкнутого по сдвигу множества L'.

Положение 2. В том случае, когда рассматриваемая модель — простая полугрупповая, множество L берется как пересечение множества $\mathcal{L}_\nu^Y$ (так обозначается множество всех ν-согласованных простых функций разметки) с замкнутым по сдвигу автоматным множеством L', где $L' \subseteq \mathcal{L}^Y$; при этом *автоматным* называется гладкое множество L', для которого существует конечный автомат, принимающий язык $\bigcup_{\mu \in L} Z(\mu)$.

В [13] приведено конструктивное описание класса автоматов, задающих замкнутые по сдвигу автоматные множества функций разметки, и все такие множества. Соображением в пользу выбора в качестве L' именно автоматных множеств служит конструктивность их задания. Уместно привести

Утверждение 3. *Множества функций разметки рассматриваемых в* [23] *моделей являются автоматными и замкнутыми по сдвигу.*

«Самодостаточность» выбора множества L' автоматным будет видна из утверждений 5 и 6. Их формулировке предпошлем необходимые понятия.

Говорим, что *схема* G_1 (ν, L)*-включает схему* G_2, если всякий раз, когда G_1 останавливается на функции из L, схема G_2 останавливается тоже, и цепочки, с которыми они останавливаются, ν-эквивалентны. Важность отношения (ν, L)-включения следует из того, что две схемы (ν, L)-эквивалентны в том и только том случае, когда каждая из них (ν, L)-включает другую. Таким образом, верно

Утверждение 4. *Из разрешимости проблемы* (ν, L)*-включения в классе схем следует разрешимость в нем проблемы* (ν, L)*-эквивалентности.*

В утверждениях 5 и 6 рассматриваются простые модели. Напомним, что их параметры обозначаются по-прежнему через ν и L, хотя теперь ν — это эквивалентность в Y^*, а L — множество простых функций разметки.

Назовем отношение эквивалентности ν *свободно коммутативным*, если ν-эквивалентны те и только те цепочки, для которых число вхождений символа y из Y в одну из них совпадает с числом его вхождений в другую, и так для любого y из Y. В переводе на требования к программам, аппроксимируемым моделью со свободно коммутативным отношением ν, получаем следующее: результат применения к любому набору данных последовательности операторов не зависит от порядка применяемых операторов.

Распределение сдвигов s называется *ортогональным*, если для любого y из Y выполнено соотношение $sy \neq \varnothing$, а для любых различных y_1 и y_2 из Y имеет место равенство $sy_1 \cap sy_2 = \varnothing$.

Из утверждения 3 следует, что множество L, с ортогональным распределением сдвигов s является автоматным и замкнутым по сдвигу. На языке требований к операторам и булевым выражениям, используемым в реальных программах, ортогональное распределение сдвигов можно трактовать следующим образом: всякое булево выражение существенно зависит хотя бы

от одной переменной из числа составляющих левые части операторов присваивания, и никакие два различных булевых выражения не имеют общих существенных переменных.

Утверждение 5. *В простой модели с параметрами ν и L, где отношение эквивалентности ν свободно коммутативно, а L совпадает с множеством $\mathscr{L}_\nu^Y$, проблема включения разрешима.*

Утверждение 6. *В простой модели, построенной над алфавитами Y и P, $|Y| = |P| \geqslant 2$, параметрами которой являются свободно коммутативная эквивалентность ν и множество L, равное $\mathscr{L}_\nu \cap L_s$, где s — ортогональное распределение сдвигов, проблема включения неразрешима.*

Утверждение 5 лéгче всего извлекается из результатов, полученных в [20]. Справедливость утверждения 6 зиждется на двух фактах: неразрешимости проблемы включения n-ленточных бинарных автоматов при $n \geqslant 2$ (см. [27]) и сводимости этой проблемы к проблеме включения в модели, рассматриваемой в утверждении 6, где n — число элементов в Y.

Наконец, сформулируем

Положение 3. В случае полугрупповой модели рекурсивных программ множество L' конструируется по заданному множеству простых функций разметки.

Примером такого конструирования являются перегородчатые модели, описываемые в § 5.

§ 5. Проблема эквивалентности схем и некоторые результаты по ее решению

Как уже отмечалось, проблема эквивалентности схем является одной из основных в проблематике теории моделей программ. Вообще обращение к схемам программ как к средству аппроксимации самих программ «вдохновлялось» положительным решением этой проблемы, нащупанным Ю. И. Яновым. Однако первые же шаги в направлении создания моделей вычислений, тоже аппроксимирующих программы и фактически обобщающих схемы Янова, привели к тому, что были обнаружены модели с неразрешимой проблемой эквивалентности. Мы имеем в виду результаты, полученные А. А. Летичевским для дискретных преобразователей [4] и М. Патерсоном для стандартных схем программ [26].

В моделях программ, введенных нами, действуют следующие ориентиры:

Теорема 5. *Существуют полугрупповые модели рекурсивных программ с неразрешимой проблемой эквивалентности.*

Теорема 6. *В максимальной модели рекурсивных программ проблема эквивалентности разрешима.*

Теорема 6 следует из излагаемой ниже теоремы 10, и это будет отмечено специально.

Цепочка рассуждений, приводящих к теореме 5, выглядит следующим образом. Прежде всего, имеет место

Утверждение 7. *Если в модели программ неразрешима проблема пустоты, то в ней неразрешима и проблема эквивалентности.*

Проблема пустоты в модели состоит в отыскании алгоритма, который для всякой схемы из этой модели устанавливает, пуста ли она в модели. *Пустота* же *схемы в модели* определяется так: схема не останавливается ни на одной функции разметки, допустимой для данной модели; другими словами, область определения схемы в данной модели пуста.

Из утверждения 7 следует, что для доказательства теоремы 5 достаточно обнаружить модель с неразрешимой проблемой пустоты.

Далее используем такой факт: для всякой простой полугрупповой модели можно построить полугрупповую же модель рекурсивных программ, подмоделью которой является первая. Поэтому достаточно найти простую полугрупповую модель с неразрешимой проблемой пустоты.

Но наличие таковой вытекает из упомянутых выше работ [4] и [26]. Дело в том, что в них были выявлены класс дискретных преобразователей и, соответственно, класс стандартных схем именно с неразрешимой проблемой пустоты. И для каждого из этих классов можно построить такую простую полугрупповую модель, что проблема пустоты в классе сводится к проблеме пустоты в построенной для него модели.

Отметим, что позднее [8] была построена простая полугрупповая модель с неразрешимой проблемой пустоты, и этот факт доказывался без ссылок на работы [4] и [26].

На основании теорем 5 и 6 появились задачи по выделению моделей с разрешимой и моделей с неразрешимой проблемой эквивалентности.

Остановимся на первой задаче и отметим некоторые из полученных результатов. Для простых моделей программ они даются теоремами 7–9.

Теорема 7. *В простой (ν, L)-модели с разрешимым отношением ν проблема эквивалентности разрешима, если ν определяет в Y^* полугруппу с левым сокращением и неразложимой единицей, а L равно $\mathscr{L}_\nu$.*

Поясним используемые здесь понятия. Говорим, что отношение ν определяет в Y^* полугруппу с левым сокращением и неразложимой единицей, если оно является полугрупповой эквивалентностью и, вдобавок к этому, для любых цепочек h_1, h_2, g из Y^*, выполняются следующие свойства.

Свойство левого сокращения: $(gh_1 \overset{\nu}{\sim} gh_2) \Longrightarrow (h_1 \overset{\nu}{\sim} h_2)$.

Свойство неразложимости единицы: $g \overset{\nu}{\not\sim} e$, где e — пустая цепочка.

Отметим, что все обсуждаемые в теореме 7 модели являются полугрупповыми.

Теорема 7 представляет собой трансляцию в модели программ результата, полученного А. А. Летичевским в [1]. Примечательность его заключается в том, что это было первое массовое положительное решение проблемы эквивалентности схем программ, правда, полученное на языке дискретных преобразователей.

В теореме 7 рассматриваются полугрупповые эквивалентности без циклов, т. е. такие отношения ν, что для любых цепочек h_1 и h_2 из Y^* выполнено соотношение

$$h_1 h_2 \overset{\nu}{\not\sim} h_1.$$

Примером отношения ν с циклами является эквивалентность, определяющая свободную полугруппу с добавлением констант. Она вводится следующим образом. Алфавит Y разбивается на два непустых непересекающихся подалфавита, Y_1 и Y_2, и требуется, чтобы для любых цепочек h_1 и h_2 из Y_1^* имело место соотношение

$$(h_1 \overset{\nu}{\sim} h_2) \Longleftrightarrow (h_1 = h_2),$$

а для любых h из Y^* и y из Y_2 — эквивалентность $hy \overset{\nu}{\sim} y$.

Символы из Y_2 называются константами. (Содержательно: оператор, обозначенный константой, осуществляет «сброс» любого состояния памяти в некоторое фиксированное.)

Очевидно, что введенная эквивалентность — полугрупповая.

Теорема 8. *В простой (ν, L)-модели, где отношение ν определяет свободную полугруппу с добавлением констант, а L равно $\mathscr{L}_\nu^Y$, проблема эквивалентности разрешима.*

Эта теорема представляет собой перенос на модели программ результата, установленного Л. П. Лисовиком в [5].

Рассматриваемые в теоремах 7 и 8 модели имеют одно общее: используемое в модели множество функций разметки равно $\mathscr{L}_\nu^Y$, т. е. полностью определяется выбором эквивалентности ν. Возможность нетривиального выхода за пределы такого выбора множества L показывает

Теорема 9. *В простой (ν, L)-модели со свободно коммутативной эквивалентностью ν и множеством L, равным $\mathscr{L}_\nu^Y \cap L'$, где автоматное и замкнутое по сдвигу множество L' определяется монотонностью операторов, проблема эквивалентности разрешима.*

Дадим терминологические пояснения. Говорим, что множество L' определяется монотонностью операторов, если оно состоит из всех таких простых функций разметки μ, что для любой цепочки h из Y^* и любого символа y из Y выполнено соотношение

$$\mu hy \geqslant \mu h;$$

при этом отношение $\geqslant$ в множестве X вводится так: для $x_1, x_2 \in X$ положим

$$(x_1 \geqslant x_2) \iff (\forall p \in P)\ (x_1(p) \geqslant x_2(p)).$$

Монотонность оператора можно трактовать следующим образом: если до выполнения оператора булево выражение было истинным, то оно таким останется и после выполнения оператора, и это верно для любого булева выражения, используемого в программе.

Теорема 9 доказана в [11].

Перейдем к рассмотрению моделей рекурсивных программ. Введем понятие перегородчатой модели.

Перегородчатая модель строится по заданной простой полугрупповой модели. Пусть τ и l — параметры последней; тогда по τ определяется эквивалентность ν в H, а по l — множество функций разметки L, где $L \subseteq \mathscr{L}$. Это и будут параметры перегородчатой модели.

Эквивалентность ν вводится следующим образом. Цепочки h_1 и h_2 из H считаем ν-эквивалентными в том и только том случае, когда совпадают их проекции на множество $C \cup R$, и, кроме того, если представить цепочки h_i, $i = 1, 2$, в виде $h_{0i} b_1 h_{1i} b_2 \ldots b_k h_{ki}$, $k \geqslant 0$, где $b_1 b_2 \ldots b_k$ есть общая проекция цепочек h_1, h_2 на множество $C \cup R$, то при всех j, $j = 0, \ldots, k$, имеет место соотношение

$$h_{j1} \overset{\nu}{\sim} h_{j2}.$$

Множество функций разметки L строится следующим образом. Представим H в виде дерева, корень которого не имеет метки, а остальные вершины помечены символами из множества $Y \cup C \cup R$. Разметка этого дерева элементами из X, определяющая некоторую функцию μ из L, осуществляется по такому правилу: для всякого вхождения в дерево символа b из $C \cup R$ рассматривается поддерево, вырастающее из вершины, несущей этот b, и соответствующее множеству Y^*, значения функции μ на этом поддереве берутся совпадающими со значениями какой-либо функции из l, причем выбор последней может зависеть от выбора символа b и его вхождения в дерево.

Легко показать, что если (τ, l)-модель — полугрупповая, то и (ν, L)-модель тоже полугрупповая. Кроме того, очевидно, что простой подмоделью построенной (ν, L)-модели является именно исходная (τ, l)-модель.

Содержательно: в перегородчатой модели составные части схемы (главный и неглавные подграфы) отделены друг от друга не только структурно, но и функционально. Под этим понимается следующая ситуация: если схемы эквивалентны, то между подграфами одной и другой имеется соответствие, сохраняющее выполняемые ими функции (при надлежащем их определении).

Теорема 10. *Проблема эквивалентности в перегородчатой модели рекурсивных программ сводится к трем проблемам в ее простой подмодели — пустоты, эквивалентности и непустоты пересечения.*

Проблема непустоты пересечения в модели состоит в отыскании алгоритма, который, получив на свой вход две произвольные схемы из этой модели, устанавливает, имеется ли из числа допустимых для данной модели функций разметки хотя бы одна такая, на которой останавливаются обе схемы.

Теорема 10 представляет собой обобщение теоремы, доказанной в [14] для частного случая перегородчатой модели: в качестве множества l простых функций разметки в [14] брались множества $\mathscr{L}_\tau^Y \cap l'$, где множество l' — автоматное и замкнутое по сдвигу.

Теорема 10 открывает путь для применения результатов, полученных для простых моделей, к моделям рекурсивных программ. Примером такого применения является сформулированная выше теорема 6: ведь простой подмоделью максимальной модели является множество схем Янова с сильной эквивалентностью в нем, и для этой подмодели разрешимы все три указанные в теореме 10 проблемы. В § 6 мы еще вернемся к обсуждению перегородчатых моделей программ.

Остальные результаты по проблеме эквивалентности в моделях рекурсивных программ имеют достаточно разрозненный характер. Они получены переносом фактов, установленных в рамках иных, чем рассматриваемое нами, определений схемы программы.

§ 6. Об одной методике поиска алгоритмов, разрешающих эквивалентность схемы

Остановимся на задаче построения алгоритмов, разрешающих эквивалентность простых моделей, которые описаны в теореме 7. Сам факт разрешимости проблемы эквивалентности в этих моделях следует из результатов, полученных в [1]. Однако предложенные там алгоритмы разрешения имеют экспоненциальную временну́ю сложность, и она сохраняется при непосредственном их переводе на случай моделей программ. Это обстоятельство заставляет искать более простые разрешающие алгоритмы.

Подсказывающим путь их поиска стал подход, которым руководствуется алгоритм Мура, распознающий эквивалентность конечных автоматов (его описание дается, например, в [25]). Наметилась определенная методика поиска. Изложим ее предписания, предполагая заданной некоторую простую модель (следовательно, задана и эквивалентность схем):

1) свести эквивалентность в множестве всех схем к эквивалентности в подмножестве так называемых *унифицированных схем* (*u-схем*) и далее рассматривать только *u*-схемы;

2) использовать понятие пары сочетаемых маршрутов в *u*-схемах, построить *критерий сочетаемости маршрутов* и убедиться в его алгоритмической проверяемости;

3) выразить отношение эквивалентности *u*-схем в требованиях, предъявляемых к парам сочетаемых маршрутов, т. е. построить *критерий эквивалентности*;

4) найти на множестве пар сочетаемых маршрутов, построенных для двух *u*-схем, такие свойства, которые присущи эквивалентным схемам (назовем их *признаками эквивалентности*; каждый признак должен быть алгоритмически проверяемым, а общее их число — конечным);

5) для любой пары схем по их форматам определить конечное, называемое *контрольным*, множество пар сочетаемых маршрутов и доказать теорему о том, что две *u*-схемы эквивалентны в том и только том случае,

когда на контрольном множестве выполнены требования критерия эквивалентности и удовлетворены признаки эквивалентности. Такую теорему назовем *финитной*.

Если удается осуществить все предписания методики и, в частности, доказать финитную теорему, то можно рассчитывать на построение алгоритма, разрешающего с полиномиальной сложностью эквивалентность схем.

Прокомментируем изложенные предписания.

Первое из них является безусловно реализуемым, причем способ реализации не зависит от выбранной модели. Это видно из самого определения *и-схемы*, т. е. схемы, структура которой удовлетворяет таким требованиям:

а) дуга из входа схемы и дуга из любого преобразователя ведет в корень дерева распознавателей (вид его представлен на рис. 3; m — число элементов в P), при этом никакие две из перечисленных дуг не ведут в один и тот же корень;

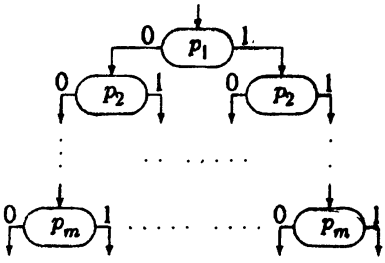


Рис. 3

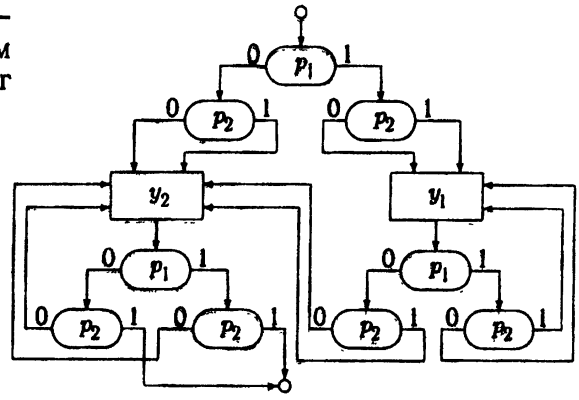


Рис. 4

б) дуга, исходящая из распознавателя, находящегося в последнем ярусе дерева, ведет либо в выход схемы, либо в преобразователь, либо в *пустой цикл*, т. е. в распознаватель с исходящими дугами, направленными в него же;

в) в схеме любой преобразователь принадлежит какому-либо пути из ее входа в выход;

г) если в схеме имеется пустой цикл, то он единственен и достижим ориентированным путем из входа, а всякий иной распознаватель содержится в каком-либо из деревьев.

Очевидно, что свойство «быть u-схемой» распознаваемо.

Утверждение 8. *Существует алгоритм, который по любой простой схеме строит строго эквивалентную ей и-схему.*

В качестве иллюстрации на рис. 4 изображена u-схема, построенная таким алгоритмом для схемы, представленной на рис. 1.

Двинемся к предписанию 2). Понятия маршрута в u-схеме и пары сочетаемых маршрутов вводятся одинаково для всех моделей.

Маршрутом в и-схеме называется путь, начинающийся в ее входе и заканчивающийся либо в преобразователе, либо в выходе схемы, либо в пустом цикле. Маршруту сопоставляется несомая им цепочка. Она строится следующим образом: при просмотре маршрута от начала к концу выписываются друг за другом символы, стоящие в принадлежащих маршруту преобразователях.

Сочетаемыми называются маршруты в u-схемах, прокладываемые общей функцией разметки из числа допустимых для данной модели.

Комментируя далее методику поиска, будем рассматривать *свободно коммутативную модель*, т. е. простую (ν, L)-модель, в которой ν — свободно коммутативная эквивалентность, а L равно $\mathcal{L}_\nu^V$. Фактически для этой модели методика поиска применена в [20], но формальное следование ее предписаниям осуществлено в [17].

Сначала опишем критерий сочетаемости маршрутов. Пусть w — маршрут в u-схеме. Для всякого его собственного подмаршрута можно говорить о том, по какой ветви дерева он продолжается в w. При этом ветвь дерева будем идентифицировать элементом x, $x \in X$, удовлетворяющим такому требованию: из любого распознавателя, принадлежащего ветви, ветвь идет по дуге, помеченной значением $x(p)$, где p — переменная, сопоставленная этому распознавателю.

Критерий сочетаемости маршрутов в свободно коммутативной модели формулируется так: маршруты w_1 и w_2 сочетаемы в том и только том случае, когда для любых равновеликих их собственных подмаршрутов из эквивалентности несомых ими цепочек следует, что эти подмаршруты продолжаются в w_1, w_2 по одинаково идентифицируемым ветвям.

Легко видеть, что этот критерий алгоритмически проверяем.

Итак, мы реализовали предписание 2). Перейдем к предписанию 3).

Критерий эквивалентности u-схем в свободно коммутативной модели выглядит следующим образом: две u-схемы эквивалентны в том и только том случае, когда любая пара их конечных, равновеликих и сочетаемых маршрутов оканчивается в вершинах общего для пары типа, и если последний — выход, то несомые маршрутами цепочки эквивалентны.

Продолжим. В рассматриваемом случае имеется один признак эквивалентности. Сформулируем его.

Упорядочим множество Y: $Y = \{y_1, y_2, \ldots, y_n\}$, $n > 0$. Всякой паре конечных маршрутов (w_1, w_2) сопоставим n-мерный вектор $\lambda(w_1, w_2)$, построенный следующим образом: его i-я компонента, $i = 1, \ldots, n$, равна разности между числом вхождений символа y_i в цепочку, несомую маршрутом w_1, и числом вхождений в цепочку, несомую маршрутом w_2.

Признак эквивалентности состоит в следующем. Пусть каждая из двух пар маршрутов, (w_1, w_2) и (w_3, w_4), ведет в общую для нее вершину схемы. Тогда если w_1 и w_3 равновелики и согласованы и w_2 и w_4 также равновелики и согласованы, то

$$\lambda(w_1, w_3) = \lambda(w_2, w_4).$$

Алгоритмическая проверяемость этого признака очевидна.

Предписание 4) реализовано. Идем дальше.

Пусть G_1 и G_2 — произвольные u-схемы, содержащие N_1 и N_2 преобразователей, соответственно. Для этих схем контрольное множество определяется состоящим из всех пар равновеликих сочетаемых маршрутов, длина которых не превосходит $m^2 N_1 N_2$, где m — число переменных в P. Финитная теорема звучит так: в свободно коммутативной модели две u-схемы эквивалентны в том и только том случае, когда на их контрольном множестве выполнено требование критерия эквивалентности и удовлетворен признак эквивалентности.

В заключение нашего комментария приведем оценку временной сложности алгоритма, распознающего эквивалентность u-схем в свободно коммутативной модели. Она выражается числом

$$O(N_1 N_2 \log(N_1, N_2)),$$

где N_1 и N_2 — количество преобразователей в u-схемах G_1 и G_2 соответственно, поступивших на вход алгоритма.

Описанная нами методика впервые применена для модели, рассматриваемой в теореме 9 (см. [10, 11]). Последние случаи ее применения — это работы [2] и [3]. В первой из них рассматривается подмножество схем рекурсивных программ со специальным образом введенной эквивалентностью схем. В этом случае разрешимость эквивалентности была установлена в [21], но способом, не подсказывающим алгоритма разрешения, который имеет полиномиальную сложность.

В [3] по существу дается решение такой задачи: описать класс простых моделей с разрешимой эквивалентностью, допускающих применение изложенной методики поиска. По построении класса моделей для некоторых из них приводятся алгоритмы разрешения, опирающиеся на эту методику. Все они имеют невысокую полиномиальную сложность. Отметим еще одно обстоятельство, побуждающее к выявлению тех случаев, когда применима описанная методика поиска. Практика (см. [17]) показала, что если верна финитная теорема, то имеет место

Утверждение 9. *Для любой пары сочетаемых маршрутов, завершающихся в выходах эквивалентных и-схем, контрольному множеству принадлежит пара сочетаемых маршрутов, заканчивающихся теми же ветвями деревьев, что и первая пара.*

Но тогда в тех специфических условиях, когда в теореме 10 рассматривается проблема непустоты пересечения, она оказывается разрешимой. К тому же из разрешимости проблемы эквивалентности следует разрешимость проблемы пустоты, а следовательно, оказывается справедливой

Теорема 11. *В перегородчатой модели рекурсивных программ проблема эквивалентности разрешима, если для ее простой подмодели верны финитная теорема и утверждение* 9.

Дополнительная важность теоремы 11 состоит в том, что она открывает дорогу для построения быстрых алгоритмов, распознающих эквивалентность в перегородчатых моделях.

§ 7. Построение полных систем эквивалентных преобразований схем программ

По традиции, начиная с работы Ю. И. Янова [23], полная система эквивалентных преобразований схем отыскивается в модели с разрешимой проблемой эквивалентности. Одним из первых требует решения вопрос о том, как задавать систему преобразований. При ответе на него в качестве основной принимается такая позиция: наше знание природы эквивалентности станет исчерпывающим, если будет ясно, как структурно отличаются одна от другой эквивалентные схемы. Согласно этой позиции, нас не удовлетворяет, например, система, состоящая из всех пар эквивалентных схем (такую систему можно предъявить сразу — ведь в рассматриваемой модели разрешима проблема эквивалентности, т. е. множество всех пар эквивалентных схем является разрешимым и определяет полную систему эквивалентных преобразований схем).

Для структурных преобразований схем используется понятие фрагмента схемы и операция замены в схеме одного ее фрагмента другим. Впервые к строгому определению понятия фрагмента и сопутствующих ему понятий обратился Ю. И. Янов [24]. Излагая ниже основанный на них способ задания системы эквивалентных преобразований схем, мы будем ориентироваться на определения, данные в [24] и повторенные в [19], учитывая их адаптацию к рассматриваемым нами схемам. При этом ограничимся содержательными описаниями.

Фрагментом схемы называется ее часть, для которой указана связь с остальной частью схемы посредством дуг, входящих во фрагмент и выходящих из фрагмента. Обычно фрагмент схемы задается без предъявления самой схемы: достаточно быть уверенным, что таковая существует. Примером фрагмента является дерево распознавателей, представленное на рис. 3. Входящая в него дуга ведет в корень дерева, выходящие из него дуги начинаются в распознавателях последнего яруса.

Для любых заданных фрагмента и схемы всегда можно определить, имеется ли вхождение первого во вторую, и найти все вхождения.

Два фрагмента называются *согласованными*, если заменой в произвольной схеме любого вхождения одного из них вхождением другого мы получим снова схему. Согласованность фрагментов — распознаваемое отношение. Например, согласованными являются фрагменты, изображенные на рис. 5. Другим примером согласованности является пара из пустого фрагмента и фрагмента, в который не входит ни одна дуга и в котором не содержится вход схемы.

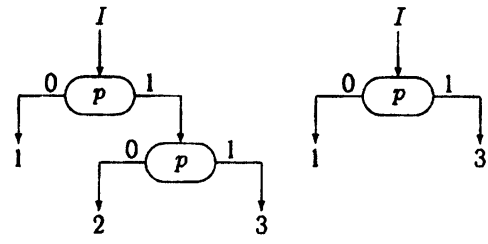


Рис. 5. Пример согласованных фрагментов. Символом I обозначена вершина, в которой начинается входящая во фрагмент дуга, а символами 1, 2, 3 — вершины, в которых оканчиваются дуги, выходящие из фрагмента

Всякая пара (F_1, F_2) согласованных фрагментов определяет множество преобразований, каждое из которых устроено так: берем произвольную схему G_1, находим какое-либо вхождение в нее фрагмента F_1, заменяем это вхождение вхождением фрагмента F_2, получаем схему G_2; о преобразовании (G_1, G_2) говорим, как об индуцированном парой (F_1, F_2). Варьируя схему G_1, вхождения в нее фрагмента F_1, а также меняя ролями фрагменты F_1 и F_2, мы получим множество преобразований $T(F_1, F_2)$. Обращаясь к фрагментам, изображенным на рис. 5, заметим, что при замене правого фрагмента левым вершина, обозначенная символом 2, может быть выбрана произвольно; так появляется еще один признак варьирования при построении множества $T(F_1, F_2)$.

В общем случае замена одного из фрагментов F_1 или F_2 другим не является безусловно допустимой. Если она разрешена при условии α, мы получим множество преобразований $T(F_1, F_2, \alpha)$. Предполагается, что для любой схемы и любого вхождения в нее фрагмента F_1 или F_2 алгоритмически распознаваемо, удовлетворяет ли это вхождение условию α. Примером условной замены является операция склеивания двух однотипных вершин. Она состоит в том, что все дуги, приходящие в одну из склеиваемых вершин, перебрасываются на другую. Эта операция применяется только при условии эквивалентности склеиваемых вершин; при этом понятие эквивалентности вершин схемы вводится по аналогии с понятием эквивалентности состояний конечного автомата, и распознаваемость эквивалентности вершин следует из разрешимости проблемы эквивалентности.

Если множество $T(F_1, F_2)$ (соответственно, множество $T(F_1, F_2, \alpha)$) состоит из эквивалентных преобразований, то фрагменты F_1 и F_2 называются *безусловно эквивалентными* (соответственно, *эквивалентными относительно* α). Так, первые две пары фрагментов, приведенные в качестве примера согласованности, являются безусловно эквивалентными. Эквивалентность вершин в схеме определяется таким образом, что операция склеивания вершин заведомо эквивалентна, если эквивалентны склеиваемые вершины.

Рассматриваются системы эквивалентных преобразований, которые конструируются следующим образом. Берется конечное число разрешимых множеств пар согласованных фрагментов. Каждое множество называется *аксиомой* и снабжается своим условием α (в частности, α может быть тождественно истинным). Требуется, чтобы каждая пара фрагментов из аксиомы была эквивалентной относительно α. В систему включаются все преобразования, индуцируемые парами фрагментов из аксиом с учетом α, и только они. Построенная система эквивалентных преобразований называется *конечной* и определяемой выбранными аксиомами.

Приведем пример системы аксиом, полной для схем Янова с сильной их эквивалентностью. Условимся при этом сам факт включения пары фрагментов в аксиому именовать отношением их равносильности. Система определяется пятью аксиомами. В каждой из них замена одного из фрагментов пары, входящей в аксиому, другим принадлежащим ей фрагментом считается безусловно допустимой, т. е. сопровождающее аксиому условие является тождественно истинным; оно просто опускается при описании аксиомы.

Заранее введем понятия, используемые в описании аксиомы 5. Пусть F — фрагмент, все вершины которого — распознаватели и в который входит единственная дуга. Будем рассматривать пути в F с попарно различными внутренними вершинами, начинающиеся входящей в F дугой и завершающиеся либо выходящей из F дугой, либо дугой, приводящей в уже пройденный ранее распознаватель. Такой путь именуем x-путем, где $x \in X$, если из всякого принадлежащего ему распознавателя он идет по дуге с меткой $x(p)$, где p — сопоставленная распознавателю переменная. Заметим, что всякому x из X соответствует x-путь, причем он единственен. Примером описанного фрагмента F является дерево распознавателей. Для всякого x в нем x-путь завершается выходящей из F дугой.

Перейдем, наконец, к описанию самих аксиом.

Аксиома 1. Фрагмент, не имеющий входящих дуг и не содержащий входа схемы, равносилен пустому фрагменту.

Аксиома 2. Фрагмент, не имеющий выходящих дуг и не содержащий выхода схемы, равносилен фрагменту, состоящему из пустого цикла, в который направлены все входящие дуги первого фрагмента.

Аксиома 3. Предположим, что фрагмент F_1 допускает разбиение множества всех его вершин на такие непересекающиеся классы $V_1, V_2, \ldots, V_k$, что:

1) каждый класс состоит либо только из преобразователей, либо только из распознавателей; всем вершинам класса сопоставлен один и тот же символ или одна и та же переменная;

2) если некоторая дуга, исходящая из вершины класса V_i, $i = 1, \ldots, k$, ведет к вершине класса V_j, $j = 1, \ldots, k$, то соответствующая ей дуга, исходящая из любой вершины класса V_i, тоже ведет к вершине класса V_j (соответствующими считаем дуги, исходящие из преобразователей, а также одинаково помеченные дуги, исходящие из распознавателей);

3) если некоторая дуга, исходящая из вершины класса V_i, $i = 1, \ldots, k$, является выходящей дугой фрагмента F_1, то соответствующие ей дуги всех других вершин класса V_i тоже являются выходящими дугами фрагмента F_1, и все они ведут в одну и ту же вершину.

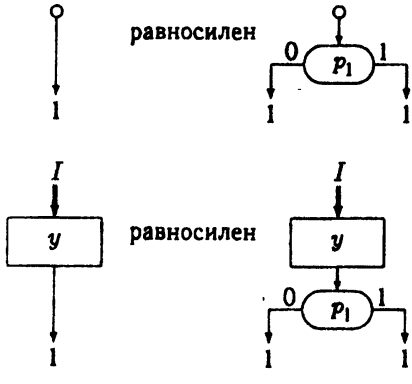


Рис. 6

Обозначим через F_2 фрагмент, получаемый из F_1 следующим построением. В каждом классе V_i выделяется одна вершина v_i, и если выходящая из нее дуга вела в вершину класса V_j, то она направляется в вершину v_j; всякая входящая дуга фрагмента F_1, оканчивающаяся в какой-либо вершине класса V_i, $i = 1, \ldots, k$, направляется в вершину v_i; после проведенных операций вершины, отличные от выделенных, опускаются.

Тогда фрагменты F_1 и F_2 равносильны.

Аксиома 4. Составляющие ее пары фрагментов представлены на рис. 6. В первой паре входящая дуга начинается во входе схемы. Во второй паре двойными стрелками изображаются множества входящих во фрагменты дуг; по-

лагаем, что между дугами одного и другого имеется взаимно однозначное соответствие, сохраняющее их начала и метки.

Аксиома 5. Пусть F_1 — фрагмент, все вершины которого — распознаватели, и в который входит единственная дуга. Тогда фрагмент F_1 равносилен фрагменту F_2, где F_2 — дерево (возможно, дополненное пустым циклом — для определенности, с переменной p_1), удовлетворяющее (для всех x из X) таким требованиям:

1) входящая в F_2 дуга начинается в той же вершине, что и дуга, входящая в F_1;

2) если x-путь в F_1 оканчивается выходящей дугой, то x-путь в F_2 ведет в ту же вершину, что и он;

3) если же x-путь в F_1 оканчивается в распознавателе, то x-путь в F_2 ведет в пустой цикл.

Приведенная нами система аксиом интересна тем, что она дает систему эквивалентных преобразований в любой простой модели.

Перейдем теперь к вопросу о том, как осуществляется поиск системы эквивалентных преобразований, полной в модели или в классе схем в ней. Существует определенная стратегия поиска. Впервые она была изложена в [9] и применялась в [17–19]. Согласно этой стратегии, сначала для модели строится алгоритм канонизации пар схем. Напомним, что такой алгоритм, получив на свой вход две произвольные схемы из данной модели, начинает процесс их эквивалентных преобразований, который завершается приведением схем к общему виду, если они эквивалентны, и прерывается на некотором шаге с объявлением о неэквивалентности схем, если таковая действительно имеет место. Как и всякий другой, алгоритм канонизации во время своей работы выполняет не только преобразовательные, но и распознавательные функции. К ним, например, относится выявление фрагмента схемы, не имеющего входящих дуг и не содержащего входа — только по выполнении этой распознавательной акции идет преобразовательная: удаление из схемы выявленного фрагмента.

Цель построения алгоритма канонизации пар схем состоит, во-первых, в том, чтобы вычленить все выполняемые им структурные преобразования схем, представить их в виде композиций действий, воспринимаемых как основные, и скомпоновать из последних искомую систему преобразований. Во-вторых, предполагается использовать сам алгоритм канонизации для доказательства полноты построенной системы.

Стратегия поиска системы эквивалентных преобразований предусматривает, из каких подалгоритмов синтезируется алгоритм канонизации пар схем. Перечислим их.

Сначала каждая схема преобразуется в эквивалентную ей u-схему. Последняя в случае схем рекурсивных программ определяется требованиями, аналогичными входящим в определение u-схемы простой программы. Различие состоит в том, что теперь роль, отводившуюся преобразователям схемы, будут играть, кроме них, вызовы и возвраты, а вместо произвольных путей из входа в выход схемы будут рассматриваться такие, которые потенциально могут претендовать на реализуемость на какой-либо функции разметки (они называются маршрутами). В таком пути вызов и соответствующий ему возврат расположены подобно открывающейся и закрывающейся скобкам в правильно построенном алгебраическом выражении. Строгое определение u-схемы дается в [15].

Преобразование схемы в эквивалентную ей u-схему выполнимо в любой модели рекурсивных программ; оно описано в [15]. Как и в случае простых схем, это преобразование осуществляется в лоне максимальной модели рекурсивных программ.

Следующий подалгоритм трансформирует произвольную u-схему в эквивалентную ей свободную u-схему. *Свободной в модели* называется схема, в которой всякий маршрут из ее входа в выход реализуем на некоторой функции разметки из числа допустимых для данной модели. Этот этап преобразований — в отличие от всегда выполнимого первого — выполним не всегда.

Осуществляющий его алгоритм, если таковой имеется, называется *алгоритмом либеризации схем*. Приведем утверждения 10 и 11, выявляющие случаи, когда алгоритм либеризации существует.

Утверждение 10. *Для простой (ν, L)-модели, где ν — эквивалентность без циклов, а множество L получено пересечением $\mathscr{L}_\nu$ с автоматным множеством L', алгоритм либеризации схем существует. Он является тождественным, если $L = \mathscr{L}_\nu$, так как в этом случае любая u-схема свободна.*

Утверждение 11. *Если для простой модели, индуцирующей перегородчатую, имеется алгоритм либеризации схем, то таковой может быть построен и для самой перегородчатой модели.*

На первых двух этапах алгоритм канонизации пар схем работает с каждой схемой отдельно. Последующие этапы характеризуются одновременной работой над обеими схемами. Третий этап состоит в преобразовании свободных u-схем в эквивалентные им свободные же u-схемы, логические графы которых изоморфны. По определению, *логический граф схемы* получается, если в ней все символы из Y заменить одним выделенным и то же самое проделать с символами из C и R. Заключительный этап действий алгоритма канонизации пар схем посвящен преобразованию схем, полученных на выходе алгоритма третьего этапа, в изоморфные друг другу. При этом сохраняются их свойства быть свободными u-схемами.

Отметим теперь ряд результатов по построению полных систем эквивалентных преобразований схем.

Теорема 12. *Для любой автоматной вариации максимальной модели рекурсивных программ существует полная система эквивалентных преобразований схем.*

Под *автоматной вариацией максимальной модели* понимается перегородчатая модель, индуцированная простой (u, L)-моделью с тождественным отношением ν и автоматным множеством L. Теорема 12 доказана в [14]. Ее фактическим обобщением является

Теорема 13. *Если в простой полугрупповой модели эквивалентность схем разрешается методикой, описанной в § 6, и при этом существует полная в ней система эквивалентных преобразований схем, то и для индуцируемой ею перегородчатой модели существует полная в ней система эквивалентных преобразований схем.*

За пределами перегородчатых моделей полная система эквивалентных преобразований схем рекурсивных программ построена только в одном случае, причем это — перенос на модель программ результата, полученного В. К. Сабельфельдом в [21].

Теорема 13 подогревает интерес к построению полных систем эквивалентных преобразований в моделях простых программ. Последним результатом в этой области является

Теорема 14. *Для простой свободно коммутативной модели существует полная в ней система эквивалентных преобразований схем.*

Такая система построена в [18] и состоит из семи аксиом. Эквивалентность в данной модели разрешается именно методикой, описанной в § 6, поэтому в данном случае применима теорема 13, и мы получаем следующий нетривиальный результат.

Теорема 15. *Для перегородчатой модели рекурсивных программ, индуцируемой простой свободно коммутативной моделью, имеется полная в ней система эквивалентных преобразований схем.*

В заключение § 7 поговорим о канонических формах представления схем. Под канонической формой понимается единственное (с точностью до изоморфизма) представление схемы в классе ее эквивалентности. Строится каноническая форма для u-схем.

Заметим, что хотя алгоритм канонизации пар схем и приводит эквивалентные схемы к общему виду, но последний не обязан быть канонической формой их представления.

Например, для схем Янова с сильной их эквивалентностью каноническая форма определяется таким требованием: в схеме отсутствуют различные и эквивалентные преобразователи. Более сложные требования предъявляются к канонической форме u-схем из максимальной модели рекурсивных программ [12]. Как одни, так и другие требования являются избыточными для алгоритма канонизации пар схем.

Эти соображения приводят к тому выводу, что построение канонической формы схемы — самостоятельная задача.

* * *

В заключение всей статьи отметим, что модели программ тесно связаны с другими моделями вычислений. Действительно, имеют место следующие факты:

сильная эквивалентность схем Янова сводится к эквивалентности конечных автоматов (впервые это было установлено в [28]);

существует простая модель, в которой эквивалентность схем сводится к эквивалентности многоленточных автоматов (для любого числа их лент);

эквивалентность в максимальной модели рекурсивных программ сводится к эквивалентности детерминированных автоматов с магазином (результат, установленный в [16]).

Из этих фактов следует связь моделей программ с формальными грамматиками — праволинейными и контекстно-свободными.

Отыскание моделей программ с неразрешимой проблемой пустоты выявило, что схемами можно протоколировать работу машин Тьюринга.

Эти связи помогают использовать при изучении моделей программ богатый арсенал средств, разработанных для других моделей вычислений.

Вместе с тем они позволяют обмениваться алгоритмами, решающими проблему эквивалентности, сравнивать системы преобразований, построенные в рамках той или другой модели, сопоставлять канонические формы изучаемых объектов.

В итоге теория моделей программ, сложившаяся вокруг проблем, намеченных в [6, 23], пополнила собой семейство математических моделей вычислений.

СПИСОК ЛИТЕРАТУРЫ

1. Глушков В. М., Летичевский А. А. Теория дискретных преобразователей // Избранные вопросы алгебры и логики. — Новосибирск: Наука, 1973. — С. 5–39.
2. Захаров В. А. Полиномиальный алгоритм разрешения проблемы эквивалентности унарных линейных рекурсивных схем программ // Тр. II Междунар. конф. «Дискретные модели в теории управляющих систем». — М.: Диалог-МГУ, 1997. — С. 26–29.
3. Захаров В. А. Быстрые алгоритмы разрешения эквивалентности операторных программ на уравновешенных шкалах // Математические вопросы кибернетики. Вып. 7. — М.: Наука, 1998. — С. 303–324.

4. Летичевский А. А. Функциональная эквивалентность дискретных преобразователей. I // Кибернетика. — 1969. — № 2. — С. 5–15.
5. Лисовик Л. П. Металинейные схемы с засылками констант // Программирование. — 1985. — № 2. — С. 29–38.
6. Ляпунов А. А. О логических схемах программ // Проблемы кибернетики. Вып. 1. — М.: Физматгиз, 1958. — С. 46–74.
7. Ляпунов А. А., Яблонский С. В. Теоретические проблемы кибернетики // Проблемы кибернетики. Вып. 9. — М.: Физматгиз, 1963. — С. 5–22.
8. Подловченко Р. И. Полугрупповые модели программ // Программирование. — 1981. — № 4. — С. 3–13.
9. Подловченко Р. И. О проблеме эквивалентных преобразований программ // Программирование. — 1986. — № 6. — С. 3–14.
10. Подловченко Р. И. Схемы программ с монотонными операторами // Программирование. — 1988. — № 6. — С. 3–12.
11. Подловченко Р. И. Разрешимость эквивалентности в множестве схем программ с монотонными и частично перестановочными операторами // Программирование. — 1990. — № 5. — С. 3–12.
12. Подловченко Р. И. Рекурсивные программы и иерархия их моделей // Программирование. — 1991. — № 6. — С. 44–51.
13. Подловченко Р. И., Аланакян Н. А. Регулярные модели программ // Программирование. — 1993. — № 4. — С. 3–11.
14. Подловченко Р. И. Разрешимость проблемы эквивалентных преобразований в специальных автоматных моделях рекурсивных программ // Докл. РАН. Математика. — 1994. — Т. 337, № 5. — С. 577–580.
15. Подловченко Р. И. Специальные перегородчато-автоматные модели рекурсивных программ // Программирование. — 1994. — № 3. — С. 3–26.
16. Подловченко Р. И. Абстрактные программы с процедурами и конечные автоматы с магазином // Интеллектуальные системы. — 1997. — Т. 2, вып. 1–4. — С. 275–295.
17. Подловченко Р. И. Алгоритм канонизации пар схем программ с перестановочными операторами // Программирование. — 1997. — № 6. — С. 3–16.
18. Подловченко Р. И. Система преобразований, полная в классе схем программ с перестановочными операторами // Программирование. — 1998. — № 2. — С. 58–67.
19. Подловченко Р. И., Айрапетяи М. Г. О построении полных систем эквивалентных преобразований схем программ // Программирование. — 1996. — № 1. — С. 3–29.
20. Подловченко Р. И., Захаров В. А. Быстрые алгоритмы распознавания эквивалентности в моделях операторных программ с коммутирующими операторами // Компьютерные аспекты в научных исследованиях и учебном процессе. — М.: Изд-во МГУ, 1996. — С. 3–8.
21. Сабельфельд В. К. О преобразованнн унарных линейных рекурсивных схем // Кибернетика. — 1975. — № 5. — С. 56–63.
22. Яблонский С. В. Основные понятия кибернетики // Проблемы кибернетики. Вып. 2. — М.: Физматгиз, 1959. — С. 7–38.
23. Янов Ю. И. О логических схемах алгоритмов // Проблемы кибернетики. Вып. 1. — М.: Физматгиз, 1958. — С. 75–127.
24. Янов Ю. И. О локальных преобразованиях схем алгоритмов // Проблемы кибернетики. Вып. 20. — М.: Наука, 1968. — С. 201–216.
25. Manna Z. Mathematical theory of computation. — New York: McGraw Hill Book Co., 1974.
26. Paterson M. S. Program schemata // Machine intelligence. V. 3. — Edinburg: Univ. Press, 1968. — P. 19–31.
27. Rabin M. O., Scott D. Finite automata and their decision problems // IBM J. of Research and Development. — 1959. — V. 3, № 2. — P. 114–125. [Рус. пер.: Рабин М. О., Скотт Д. Конечные автоматы и задачи их разрешения // Кибернетич. сб. Вып. 4. — М.: ИЛ, 1962. — С. 58–91.]
28. Rutledge J. D. On Ianov's program schemata // J. of ACM. — 1964. — V. 11, № 1. — P. 1–9.

Поступило в редакцию 23 IV 1998