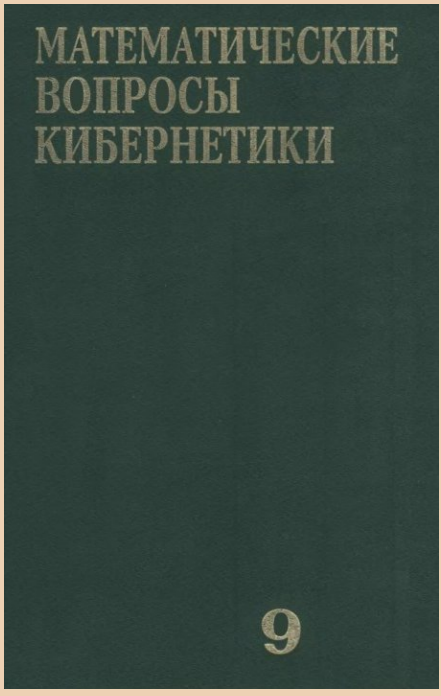
ИПМ им. М.В. Келдыша РАН • Электронная библиотека
Математические вопросы кибернетики • Выпуск 9



Р. И. Подловченко

К вопросу об эквивалентных преобразованиях алгоритмов и программ

Рекомендуемая форма библиографической ссылки: Подловченко Р. И. К вопросу об эквивалентных преобразованиях алгоритмов и программ // Математические вопросы кибернетики. Вып. 9. — М.: Физматлит, 2000. — С. 25–36. URL: http://library.keldysh.ru/mvk.asp?id=2000-25

К ВОПРОСУ ОБ ЭКВИВАЛЕНТНЫХ ПРЕОБРАЗОВАНИЯХ АЛГОРИТМОВ И ПРОГРАММ *)

Р. И. ПОДЛОВЧЕНКО

(МОСКВА)

Статья носит методологический характер и посвящена изложению концепций, сложившихся при построении эквивалентных преобразований алгоритмов и программ. Стимулом к этому послужила следующая ситуация: излагаемые концепции во многом определили общий подход к построению эквивалентных преобразований в других моделях вычислений.

Одновременное обращение к алгоритмам и программам обусловлено тем, что нами имеются в виду исключительно последовательные программы, а они представляют собой специфические записи алгоритмов. Эту суть программ впервые в отечественной литературе выразил А. А. Ляпунов [7], обращаясь к задаче формализации понятия программы. Существующие к тому времени формализации понятия алгоритма (машины Тьюринга, нормальные алгоритмы Маркова и др.) были нацелены на исследование природы вычислений, а не на практическое их использование. Предложенное А. А. Ляпуновым понятие программы зафиксировало новый подход к формализации алгоритма, учитывающий практическое его применение. Это дает основание, рассуждая об алгоритмах, иметь в виду и программы, называя их, когда требуется, программными алгоритмами.

Статья состоит из двух частей. В первой формулируется задача построения полной системы эквивалентных преобразований алгоритмов. Кратко характеризуется ситуация с ее решением, примиряющая с поиском неполных систем эквивалентных преобразований. Так мы приходим к теории схем программ, в рамках которой сформировался цельный аппарат по построению неполных систем эквивалентных преобразований для программных алгоритмов. Излагаются концепции, на которых основан этот аппарат. В числе их — признание в качестве фундаментальной проблемы эквивалентных преобразований схем, состоящей в поиске полных систем эквивалентных преобразований схем программ. Особое внимание уделяется применяемым средствам решения этой проблемы. Описывается эволюция понятия допустимого преобразования, произошедшая в теории схем программ.

Вторая часть статьи призвана проиллюстрировать, как в конкретном случае реализуются общие положения, высказанные при описании средств решения проблемы эквивалентных преобразований схем. Нами взяты схемы программ, известные в теории под названием схем Янова. На их материале конкретизируются отдельные понятия, для них строится система

*) Работа поддержана грантом Российского фонда фундаментальных исследований (проект 00-01-00133) и грантом по фундаментальным исследованиям в области математики Министерства образования РФ (проект 40).

эквивалентных преобразований и доказывается ее полнота. Отметим, что устанавливаемые здесь факты не являются новыми — они выполняют чисто иллюстративные функции. Для цельности восприятия излагаемого материала даются все используемые в нем понятия.

I

Эквивалентными преобразованиями алгоритмов называются преобразования, сохраняющие исследуемое отношение эквивалентности. Содержательное представление об эквивалентных преобразованиях прослеживается в математике с древнейших времен. Умение решать некоторые задачи в общем виде всегда сопровождалось стремлением улучшить первоначально найденное решение, и это осуществлялось применением эквивалентных преобразований. Оптимизация алгоритмов по тем или иным их характеристикам и поныне составляет важнейшую область применения эквивалентных преобразований.

Уточнение понятия эквивалентного преобразования основано на формализации рассматриваемых алгоритмов и отношения их эквивалентности.

При формализации алгоритмы предстают в виде конструктивных объектов — слов в фиксированном алфавите, конечных графов с размеченными вершинами и дугами, возможно, в виде объектов более сложной структуры. Их семантическая трактовка связана с универсальной, в смысле приложимости к произвольному объекту заданного типа, процедурой его выполнения. Посредством ее с каждым объектом ассоциируется реализуемая им функция, в общем случае, частичная. Основным из изучаемых отношений эквивалентности является отношение функциональной эквивалентности, которое определяется требованием совпадения реализуемых объектами функций. Используются и другие отношения эквивалентности, в частности, более строгие, чем функциональная эквивалентность, учитывающие не только конечный результат выполнения объекта, но и некоторую историю его получения [2].

Фундаментальной считается задача построения системы эквивалентных преобразований, полной в заданном классе K однотипных объектов. Ее постановка выглядит следующим образом. Под преобразованием понимается упорядоченная пара объектов из K; первый из них называется преобразуемым, второй — преобразованным. Если эти объекты эквивалентны (по изучаемому отношению эквивалентности), то преобразование называется эквивалентным. Совокупность преобразований именуется системой. Пусть $G_1, G_2 \in K$; говорим, что G_1 преобразуется в G_2 средствами заданной системы, если существует конечная последовательность принадлежащих ей преобразований, в которой первая пара начинается G_1, последняя оканчивается G_2, и для всех промежуточных пар выполняется требование: ее преобразуемый объект совпадает с преобразованным в предшествующей паре, а ее преобразованный объект — с преобразуемым в последующей паре. Система эквивалентных преобразований называется полной в классе K, если, какой бы ни была пара эквивалентных объектов из K, первый из них преобразуем во второй средствами этой системы. Тривиальным решением рассматриваемой задачи является система, состоящая из всех пар эквивалентных объектов из K. Обычно такое решение отвергается заранее, и отыскиваются менее мощные системы.

Практический интерес представляют разрешимые полные системы эквивалентных преобразований. Однако для класса алгоритмов, реализующих все вычислимые функции, отношение функциональной эквивалентности не является рекурсивно перечислимым, т. е. не существует разрешимой

полной системы эквивалентных преобразований. В связи с этим либо сужают класс алгоритмов в целях получения решения задачи, либо довольствуются построением неполных систем эквивалентных преобразований. Используется и другой путь: понятие полной системы эквивалентных преобразований обобщается до понятия предельно полной системы [18]. Средствами последней любые два эквивалентных алгоритма из заданного класса можно в пределе привести к общему вычислительному комплексу, возможно, бесконечному. Для класса программ, вычисляющих конечные функции, предельно полная система является полной в обычном смысле. Отметим, что теоретический интерес к данному направлению исследований превалирует над практическим.

Переход от функционально богатых классов алгоритмов к их подклассам осуществляется различными способами. В одних случаях используются автоматы — конечные, многоленточные, с магазинной памятью и прочие, в других — подкласс алгоритмов строится путем замыкания некоторого базисного множества функций посредством фиксированных операций над ними. Однако отношение функциональной эквивалентности оказывается разрешимым лишь в случаях, когда построенный подкласс функционально беден.

Поэтому исследования устремились в русло построения неполных систем эквивалентных преобразований. Здесь основными объектами стали программные алгоритмы, а основным приемом исследований — замена их моделями, именуемыми схемами программ. При всем многообразии переходов от программ к их схемам им присущи следующие общие черты:

1) исходным служит формальное определение программы — ее структуры и функционирования; на основе функционирования программе приписывается реализуемая ею функция;

2) в схеме программы сохраняется структура моделируемой ею программы; это обеспечивает положение: всякое преобразование схемы является одновременно и преобразованием программы;

3) эквивалентность схем вводится как аппроксимирующая функциональную эквивалентность программ, т. е. из первой всегда следует вторая; этим достигается ситуация: всякое эквивалентное преобразование схем одновременно является и эквивалентным преобразованием моделируемых ими программ;

4) фундаментальной объявляется задача построения системы эквивалентных преобразований, полной в изучаемом классе схем (проблема эквивалентных преобразований схем); не будучи полной для программ, моделируемых этими схемами, такая система является наиболее богатой (хотя, в общем случае, не единственной) из систем, которые можно получить в рамках выбранного класса схем;

5) отыскиваются полные системы эквивалентных преобразований схем, являющиеся разрешимыми;

6) на одно из первых мест выдвигается проблема эквивалентности схем — поиск алгоритма, который, получив на свой вход две произвольные схемы из заданного класса, устанавливает, эквивалентны они или нет;

7) средства решения проблемы эквивалентных преобразований схем ограничиваются построением формального исчисления, формулами которого являются пары фрагментов схем, единственным правилом вывода — замена в схеме вхождения одного из фрагментов пары другим, а аксиомами — разрешимые множества пар фрагментов, гарантирующих, что при применении правила вывода эти пары индуцируют эквивалентные преобразования схем.

Теория, в которой объектами служат схемы программ, исследуются аппроксимирующие отношения эквивалентности схем и основной в проблематике является проблема эквивалентных преобразований схем, называется

теорией схем программ. Ее возникновение связано с работами А. А. Ляпунова [7] и Ю. И. Янова [16]. Разделы теории определяются типами моделируемых программных алгоритмов и изучаемым отношением эквивалентности [2, 5, 12]. В качестве моделируемых выступают алгоритмы, описываемые процедурными (императивными), функциональными и логическими программами [9].

Как и следовало ожидать, среди допустимых отношений эквивалентности схем были обнаружены не являющиеся рекурсивно перечислимыми [6, 20]. И так как проблема эквивалентных преобразований схем рассматривается в классах схем с разрешимой проблемой эквивалентности, то исследование второй, как правило, предшествует исследованию первой.

Специально остановимся на средствах решения проблемы эквивалентных преобразований схем. Понятие фрагмента вводится таким образом, что схема является частным случаем фрагмента, а фрагмент схемы может рассматриваться без предъявления самой схемы. Далее вводится симметричное отношение согласованности фрагментов. Согласно ему, если F_1, F_2 — такая пара, то, какими бы ни были схема (из заданного класса) и вхождение в нее фрагмента F_i, $i = 1, 2$, всегда осуществима замена этого вхождения фрагментом F_{3-i}, и всякая замена переводит данную схему в объект, непременно являющийся схемой. Таким образом, согласованная пара F_1, F_2 индуцирует множество $T(F_1, F_2)$ преобразований схем. В общем случае формулируются условия, предъявляемые ко вхождению фрагментов F_1, F_2 в схему и выделяющие в $T(F_1, F_2)$ подмножество, состоящее из эквивалентных преобразований. Пара F_1, F_2 вместе с этими условиями называется правилом преобразований. Если условия отсутствуют, т. е. все преобразования из $T(F_1, F_2)$ являются эквивалентными, то правило F_1, F_2 именуется безусловным. Рассматриваются системы эквивалентных преобразований схем, полные или неполные, индуцируемые разрешимыми множествами правил преобразований.

Традиционным стал подход, когда такое множество задается конечным набором схем правил. Под схемой правил понимается разрешимое множество правил, полученное путем варьирования в правиле параметров фрагментов. Если варьированию подвергаются только наименования символов, используемых во фрагментах, то мы имеем дело со схемой локальных правил. В общем случае допускается варьирование и формата фрагментов. Конечный набор схем правил, индуцирующих полную систему эквивалентных преобразований, сам называется полным.

Построение системы схем правил обычно осуществляется следующим образом. Рассматриваются множества, состоящие из пар согласованных фрагментов; они именуются аксиомами. Доказывается, что каждая аксиома — это схема правил, и система определяется перечнем составляющих ее аксиом.

Таким образом, фактически строится исчисление, о котором говорилось в п. 7). При полноте системы аксиом в этом исчислении из всякой схемы программ выводима любая эквивалентная ей схема.

При поиске таких наборов возможны ограничения, налагаемые на типы правил. Часто применяемым является требование: рассматривать только схемы локальных правил. Оно восходит к случаям, когда преобразуемые объекты записываются формулами, а фрагментами являются подформулы. Такая ситуация имеет место, например, для регулярных выражений. Доказано, что для них не существует конечного полного набора схем локальных и безусловных правил [17], если же снять требование локальности, то такой набор построить можно.

Распрощаться с обременительным ограничением на тип используемых правил, выражающимся в требовании их локальности, позволил переход от формульной записи алгоритма к графовой. Для схем программ графовая запись была предложена Л. А. Калужниным [4], а первый опыт ее использования при построении конечных полных систем схем правил принадлежит А. П. Ершову [1]. Предложенные в [1] системы состоят из локальных и условных правил. Последние опираются на инварианты схем программ, построенные путем дополнительной разметки графа, определяющего схему. Разметка осуществляется применением специальных правил, которые включаются в систему. Показательно то, что этими правилами обеспечивается и снятие разметки после ее использования. Таким образом, восстанавливается объект, имеющий право называться схемой программы.

Приемы, открытые А. П. Ершовым в [1], использовались в других работах (см., например, [15]). Однако обнаружились трудности, связанные со снятием дополнительной разметки. Это обстоятельство заставило задуматься над тем, целесообразно ли включать в систему правила, осуществляющие разметку схемы программы. Ход рассуждений здесь следующий. Цель построения цепочки преобразований, трансформирующих одну из эквивалентных схем в другую, — выявить, как структура первой переходит в структуру второй. А вычисление инвариантов схемы, осуществляемое правилами разметки, оставляет структуру схемы неизменной. Отсюда решение: вычисление инвариантов схемы проводить за пределами создаваемой системы преобразований, потребовав лишь его алгоритмическую осуществимость.

Такое решение ознаменовало новый скачок в эволюции подхода к проблеме эквивалентных преобразований. С его принятием существенно продвинулось решение самой проблемы (см. [11, 13]). Вместе с тем, методы вычисления для схем программ ее инвариантов различного типа влились в новый раздел математической теории вычислений — потоковый анализ графов.

Так как в системах эквивалентных преобразований схем, индуцируемых множествами правил преобразований, всякое преобразование является обратимым, то полнота системы доказывается путем трансформации любых двух эквивалентных схем к общему виду. Она осуществляется алгоритмом, распознающим эквивалентность схем, и опирается на предварительный анализ структуры эквивалентных схем. Из-за разнотипности применяемых правил и необходимости выявлять ситуации, в которых они применяются, сложность такого алгоритма оценивать трудно, и это направление исследований разработано слабо. Вместе с тем, в последние годы активно развиваются исследования по построению быстрых алгоритмов, распознающих эквивалентность схем программ (см. [3, 14]). Они стимулируются применением распознающих алгоритмов при трансформации схем к общему виду.

Родственной по отношению к задаче построения полной системы эквивалентных преобразований алгоритмов является задача канонизации алгоритма, т. е. эквивалентного его преобразования к виду, которым идентифицируется класс эквивалентности данного алгоритма. Эта задача также переводится на схемный уровень и там решается методами, близкими к описанным выше [10].

Наряду с полными системами эквивалентных преобразований схем конструируются и неполные. Для их задания тоже используются конечные наборы схем правил. Полные и неполные системы эквивалентных преобразований схем программ активно применяются при трансляции программ с одного языка на другой и при автоматическом синтезе программ.

II

Для иллюстрации понятий и приемов, изложенных при описании средств решения проблемы эквивалентных преобразований схем, обратимся к конкретному материалу — схемам простых программ с сильной их эквивалентностью.

Сначала опишем их содержательно. Простые программы строятся над конечным базисом, состоящим из операторов и булевых выражений, применением всех средств композиции операторов, кроме аппарата процедур. Схема простой программы по своей структуре совпадает с самой программой, лишь базисные операторы и булевы выражения заменяются операторными символами и соответственно логическими переменными. Выполнение схемы сопровождается построением цепочки операторных символов. Две схемы объявляются сильно эквивалентными, если любые согласованные их выполнения либо завершаются с равными цепочками, либо не завершаются вообще.

Дадим строгие определения.

Схемы программ строятся над двумя конечными, непересекающимися алфавитами — Y и P. Элементы Y называются *операторными символами*, элементы P — *логическими переменными*; каждая из них принимает значения из множества $\{0, 1\}$. Далее алфавиты Y и P остаются неизменными и поэтому не упоминаются в наименованиях зависящих от них понятий и конструкций.

Схема программы (просто: схема) задается конечным ориентированным графом. В нем выделены две вершины: *вход* — вершина без входящих в нее дуг и с единственной исходящей дугой и *выход* — вершина без исходящих из нее дуг. Остальные вершины графа по своему типу делятся на *преобразователи* и *распознаватели*. Из каждого преобразователя исходит одна дуга, и ему сопоставлен символ из Y. Из каждого распознавателя исходят две дуги, несущие метки 0 и 1 соответственно, и ему сопоставлена логическая переменная из P.

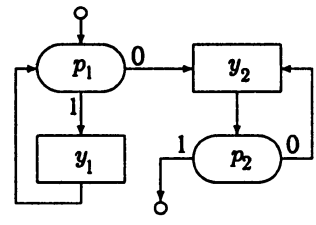


Рис. 1

Пример схемы программы приведен на рис. 1. Она представляет собой композицию двух циклов, одного — с предусловием p_1, а второго — с постусловием p_2.

Функциональное описание схемы связано с процессом ее выполнения. Последний состоит в путешествии по схеме, сопровождаемом накоплением цепочки операторных символов. Выбор пути определяется заданной apriori функцией разметки. Введем ее. Пусть

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

Элементы X интерпретируются как наборы значений всех логических переменных. Слова в алфавите Y будем называть *операторными цепочками*. *Функцией разметки* назовем отображение множества Y^*, состоящего из всех операторных цепочек, в множество X. Обозначим $\mathcal{L}$ множество всех функций разметки.

Пусть G — схема, и μ — функция из $\mathcal{L}$. Процесс выполнения схемы G *на функции* μ начинается в ее входе при пустой операторной цепочке и состоит в обходе схемы. Прохождение через преобразователь сопровождается приписыванием справа к текущей цепочке операторного символа, сопоставленного этому преобразователю. Переход через распознаватель не изменяет текущей цепочки; если последняя — это h, а распознавателю сопоставлена переменная p, то из набора μh выделяется значение переменной p, и дальнейший обход схемы продолжается по дуге, помеченной этим значением.

Выполнение схемы завершается при достижении ее выхода. В этом случае говорим, что схема *остановилась на* μ, и *результатом ее выполнения* считаем накопленную к моменту остановки операторную цепочку.

Сильная эквивалентность схем определяется следующим образом: две схемы называются сильно эквивалентными, если, какой бы ни была функция разметки из $\mathcal{L}$, всякий раз, как на ней останавливается одна из схем, останавливается и другая, и результаты их выполнения — это совпадающие операторные цепочки.

Разрешимость сильной эквивалентности в множестве всех схем над базисом Y, P — факт, давно установленный [5].

Дадим определения фрагмента схемы и сопутствующих ему понятий.

Фрагментом схемы называется ее часть, определяемая заданным множеством вершин схемы и содержащая вместе с этими вершинами все инцидентные им дуги. Вершины заданного множества именуются принадлежащими фрагменту или просто вершинами фрагмента. Они сохраняют приписанные им в схеме символы и переменные. Принадлежащие фрагменту дуги тоже наследуют из схемы свои метки. Дугу фрагмента, начинающуюся в вершине, ему не принадлежащей, назовем *входящей*, ее начало — *входом фрагмента*. Дугу фрагмента, которая ведет в вершину, ему не принадлежащую, назовем *выходящей*, ее конец — *выходом фрагмента*.

Фрагмент, определенный пустым множеством вершин, называется пустым.

Как мы видим, схема является частным случаем ее фрагмента. Обычно фрагмент задается без предъявления самой схемы, ибо всегда можно определить, допускает ли он достройку до схемы.

Назовем *изоморфными* фрагменты, которые отличаются только наименованиями своих вершин, входов и выходов и совпадают при совмещении их наименований.

Вхождением фрагмента F *в схему* назовем любой ее фрагмент, изоморфный фрагменту F.

Согласованием двух фрагментов назовем операцию, применимую к фрагментам с равновеликими множествами входов и состоящую в следующем. Между входами одного и другого устанавливается взаимно однозначное соответствие, сохраняющее тип входа и удовлетворяющее требованию: если соответствующие друг другу входы — это распознаватели, то при любом ε, $\varepsilon \in \{0, 1\}$, исходящие из них дуги с меткой ε либо обе являются входящими во фрагменты, либо обе таковыми не являются. В множестве выходов одного и другого выделяются равновеликие подмножества (возможно, пустые), и между выделенными выходами одного и другого устанавливается взаимно однозначное соответствие. Результатом описанной операции являются *согласованные фрагменты*. Отметим, что согласование фрагментов — это коммутативная операция.

Пусть F_1, F_2 — согласованные фрагменты, G — некоторая схема. Под *операцией замены в схеме* G *фрагмента* F_1 *фрагментом* F_2 понимаем следующее преобразование схемы G в схему G'. Если F_1 не имеет вхождений в G, то $G' = G$. Пусть F_1' — какое-либо вхождение фрагмента F_1 в схему G. Тогда заменим F_2 изоморфным ему фрагментом F_2', не имеющим общих вершин со схемой G, включая входы и выходы F_2'. На фрагменты F_1' и F_2' перенесем соответствие входов и выделенных выходов, введенное при согласовании F_1, F_2. Затем выполним такие действия: всякую вершину схемы G, являющуюся входом или выделенным выходом фрагмента F_1', совместим с соответствующей ей вершиной фрагмента F_2'; удалим после этого все вершины фрагмента F_1' вместе с инцидентными им дугами; наконец, невыделенные выходы фрагмента F_2' совместим с любыми из оставшихся вершин схемы G, отличными от ее входа. Легко убедиться в том, что результатом выполненных действий является схема. Примем ее за G'.

Говорим, что преобразование (G, G') индуцировано парой (F_1, F_2) и определяется выделенным вхождением фрагмента F_1 в схему G.

Как было показано в части I, полнота системы преобразований, задаваемой набором схем правил, доказывается не приведением одной из двух эквивалентных схем к другой, а трансформацией обеих схем к общему виду. Опишем алгоритм, осуществляющий такую трансформацию в нашем случае.

Пусть заданы схемы G_1^0, G_2^0. Прежде всего, каждая из них преобразуется в так называемую u-схему. В последней унифицированы переходы по распознавателям от вершин типа вход, преобразователь к вершинам типа преобразователь, выход, пустой цикл (так называется распознаватель, обе дуги которого ведут в него же). Кроме того, в u-схеме отсутствуют фрагменты, исключая образованный пустым циклом, через которые не проходят пути из входа схемы в ее выход.

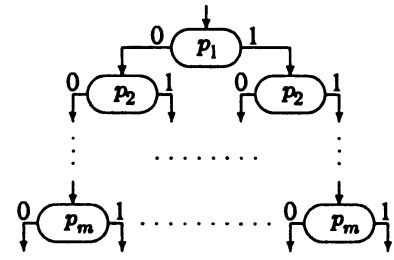


Рис. 2

Строгое определение u-схемы таково: это — схема, структура которой удовлетворяет требованиям:

1) дуга из входа схемы и дуга из любого преобразователя схемы ведет в корень дерева распознавателей, изображенного на рис. 2; при этом никакие две из перечисленных дуг не ведут в один и тот же корень;

2) всякий распознаватель схемы либо содержится в одном из указанных деревьев, либо является пустым циклом; в схеме не более одного пустого цикла;

3) дуга, исходящая из распознавателя, находящегося в последнем ярусе дерева, ведет либо в выход схемы, либо в преобразователь, либо в пустой цикл;

4) любой преобразователь схемы принадлежит какому-либо пути из ее входа в выход.

Очевидно, что свойство схемы быть u-схемой алгоритмически распознаваемо.

Отметим, что в дереве распознавателей, изображенном на рис. 2, каждому элементу x из X соответствует единственная ветвь, обладающая свойством: каким бы ни был расположенный на ветви распознаватель, если p — приписанная ему переменная, то из распознавателя ветвь идет по дуге с меткой $x(p)$; эта ветвь называется x-ветвью.

Имеет место

Т е о р е м а 1. *Существует алгоритм, который трансформирует произвольную схему в сильно эквивалентную ей u-схему.*

Поскольку такая трансформация является непременным этапом при приведении эквивалентных схем к общему виду даже в случаях, когда эквивалентность не является сильной, остановимся на описании алгоритма, существование которого утверждается теоремой 1.

Применяя его, мы выполняем такую последовательность действий:

1) добиваемся того, чтобы дуга из входа схемы и любого ее преобразователя вела непременно в распознаватель;

2) фрагмент, индуцированный всеми распознавателями схемы, расклеиваем на копии по числу входящих во фрагмент дуг, добиваясь того, чтобы каждая копия имела в точности одну входящую дугу;

3) каждую копию заменяем равноценным ей деревом распознавателей;

4) удаляем все преобразователи, не лежащие на путях из входа схемы в ее выход, вместе с произрастающими из них деревьями распознавателей;

5) склеиваем в один все пустые циклы.

Доказательство того, что каждое из этих действий можно реализовать так, что оно будет трансформировать схему в сильно эквивалентную ей, опускаем. Сам алгоритм обозначим R_1.

Пусть G_i — u-схема, построенная алгоритмом R_1 для схемы G_i^0, $i=1,2$. Далее схемы G_1, G_2 поступают на вход алгоритма R_2, который берется установить, эквивалентны они или нет. Опишем работу алгоритма R_2.

Алгоритм R_2 строит для поступающих на его вход схем таблицу, насчитывающую, кроме ведущего столбца, еще $|X|$ столбцов, поименованных различными элементами из X. В таблицу вносятся пары вида (v_1, v_2), где v_i — вершина схемы G_i, $i=1,2$, имеющая тип: вход, преобразователь, выход, пустой цикл. Заполнение таблицы ведется по строкам, содержащим элемент в ведущем столбце. Первоначально в ведущий столбец первой строки помещается пара (v_1^0, v_2^0), где v_i^0 — вход схемы G_i, $i=1,2$, и заполнению подлежит первая свободная позиция строки. Предположим, что мы продвинулись до строки с парой (v_1, v_2) в ведущем столбце, и требуется заполнить позицию в столбце этой строки, поименованном x. Как будет видно из дальнейшего, v_1, v_2 — это либо входы схем, либо преобразователи, которым приписан общий операторный символ. В рассматриваемую позицию вносится пара (v_{1x}, v_{2x}), где v_{ix} — вершина, в которую ведет x-ветвь дерева распознавателей, произрастающего из v_i, $i=1,2$. Если v_{1x}, v_{2x} однотипны, а в случае, когда они — преобразователи, им сопоставлен общий символ, то пара (v_{1x}, v_{2x}) называется благополучной. При первой же встрече с неблагополучной парой алгоритм R_2 прекращает работу, объявляя схемы G_1, G_2 не сильно эквивалентными. В случае, когда v_{1x}, v_{2x} — преобразователи с общим символом, алгоритм проверяет, содержится ли эта пара в ведущем столбце. Если она отсутствует, то алгоритм вносит ее на первую свободную позицию ведущего столбца. Далее алгоритм поступает так. Если благополучная пара (v_{1x}, v_{2x}) занимает позицию в последнем столбце последней строки с элементом в ведущем столбце, то R_2 останавливается, объявляя схемы G_1, G_2 сильно эквивалентными. В противном случае он определяет позицию таблицы, подлежащую заполнению на следующем шаге: это — позиция данной строки, идущая следом за рассматриваемой, а если таковой нет, то первая свободная позиция следующей строки. Алгоритм R_2 описан. Очевидно, что он заканчивает свою работу всегда за конечное число шагов.

Справедлива

Теорема 2. *Алгоритм R_2 распознает сильную эквивалентность в множестве u-схем.*

Доказательство ее опускаем.

Следствие теоремы 2. *Если u-схемы, поступившие на вход алгоритма R_2, сильно эквивалентны, то построенная им таблица задает u-схему, сильно эквивалентную поступившим.*

В итоге: чтобы две сильно эквивалентные схемы привести к общей и сильно эквивалентной им схеме, достаточно каждую из них алгоритмом R_1 преобразовать в u-схему, и затем для полученных схем взять схему, построенную алгоритмом R_2.

Опишем теперь предлагаемые нами аксиомы. Согласованные фрагменты, составляющие пару из аксиомы, условимся называть равносильными.

Заранее введем понятия, используемые в описании аксиомы А5.

Пусть F — фрагмент, все вершины которого — распознаватели, и входящая в него дуга единственна. 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, и он единствен. Примером описанного F является дерево распознавателей. Для всякого x в нем x-путь — это x-ветвь, и завершается он выходящей из F дугой.

Перейдем, наконец, к описанию самих аксиом.

Аксиома А1. Фрагмент, не имеющий входящих дуг и не содержащий входа схемы, равносилен пустому фрагменту.

Аксиома А2. 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 А3. Предположим, что фрагмент F_1 допускает разбиение множества всех его вершин на непересекающиеся классы $V_1, V_2, \ldots, V_k$ такие, 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.

Обозначим F_2 фрагмент, который получается 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.

Фрагменты F_1 и F_2 равносильны.

Аксиома А4. Составляющие ее пары фрагментов представлены на рис. 3. В первой паре входящая дуга начинается во входе схемы. Во второй паре двойными стрелками изображаются множества входящих во фрагменты дуг; полагаем, что между дугами одного и другого имеется взаимно однозначное соответствие, сохраняющее их начала и метки.

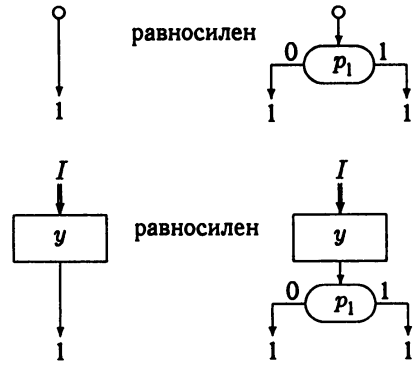


Рис. 3

Аксиома А5. Пусть F_1 — фрагмент, все вершины которого — распознаватели, и входящая в него дуга единственна. Фрагмент F_1 равносилен фрагменту F_2, где F_2 — дерево, возможно, дополненное пустым циклом (для определенности, с переменной p_1) и удовлетворяющее требованиям: входящая в F_2 дуга начинается в той же вершине, что и дуга, входящая в F_1; если x-путь в F_1 оканчивается выходящей дугой, то x-путь в F_2 ведет в ту же вершину, что и он; если же x-путь в F_1 оканчивается в распознавателе, то x-путь в F_2 ведет в пустой цикл; и так для всех x из X.

Разрешимость каждого множества, представленного аксиомами А1–А5, очевидна. Легко устанавливается (доказательство опускаем) и теорема 3.

Теорема 3. *Каждая из аксиом* А1–А5 *является схемой правил.*

Отметим, что правила всех пяти аксиом являются безусловными; правила из аксиомы А4 локальны, а все остальные таковыми не являются.

Заключаем данную часть статьи теоремой 4.

Теорема 4. *Система преобразований схем программ, определенная аксиомами* А1–А5, *полна в классе всех схем над базисом* Y, P.

Доказательство (в общих чертах). Сначала покажем, что действия, выполняемые алгоритмом R_1, реализуемы преобразованиями, индуцированными правилами из аксиом А1–А5 (для краткости будем говорить далее просто о преобразовании по аксиоме такой-то).

Очевидно, что действие 1) осуществляется преобразованиями по аксиоме А4, действие 2) — преобразованиями по аксиоме А3, действие 3) — преобразованиями по аксиоме А5. При выполнении действия 4) предварительно вычисляются два инварианта для каждого преобразователя схемы: достижимость вершины из входа схемы и достижимость из вершины выхода схемы. Легко видеть, что их вычисление алгоритмически осуществимо. После этого проводятся преобразования сначала по аксиоме А1, затем по аксиоме А2.

Итак, преобразованиями из предложенной системы исходные схемы, скажем, G_1^0, G_2^0, можно трансформировать в сильно эквивалентные им схемы; обозначим их G_1, G_2.

Теперь используем предполагаемую сильную эквивалентность схем G_1, G_2. По следствию теоремы 2 алгоритм R_2 может построить для них сильно эквивалентную им схему; обозначим ее G_{12}. Покажем, что схема G_{12} может быть преобразована в схему G_1 преобразованиями по аксиомам А1 и А3. Отсюда, ввиду обратимости каждого преобразования, принадлежащего системе, будет следовать: схему G_{12} можно получить из схемы G_1 преобразованиями по аксиомам А1, А3. Понятно, что в этих рассуждениях место схемы G_1 может занять схема G_2.

Обозначим F_1 фрагмент без входящих дуг, полученный из схемы G_1 удалением ее входа вместе с произрастающим из него деревом распознавателей, и пусть F_{12} — фрагмент, полученный из схемы G_{12} аналогичной операцией, но с сохранением всех входящих в него дуг. Используя аксиому А1, присоединим к схеме G_{12} фрагмент F_1 путем совмещения выходов. В полученной схеме рассмотрим фрагмент Ф, определенный вершинами фрагментов F_{12} и F_1. Руководствуясь указаниями, содержащимися в аксиоме А3, проведем разбивку вершин фрагмента Ф на классы таким образом, чтобы в каждом классе содержалась в точности одна вершина из F_1. Применяя преобразование по аксиоме А3, перебросим входящие дуги фрагмента F_{12} на вершины фрагмента F_1. После этого удалим из схемы оставшийся без входящих дуг фрагмент F_{12} (преобразованием по аксиоме А1). Схема G_1 получена.

Теорема 4 доказана.

СПИСОК ЛИТЕРАТУРЫ

1. Ершов А. П. Об операторных схемах Янова // Проблемы кибернетики. Вып. 20. — М.: Наука, 1968. — С. 181–200.
2. Ершов А. П. Современное состояние теории схем программ // Проблемы кибернетики. Вып. 27. — М.: Наука, 1973. — С. 87–110.
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. Калужнин Л. А. Об алгоритмизации математических задач // Проблемы кибернетики. Вып. 2. — М.: Физматгиз, 1959. — С. 51–68.
5. Котов В. Е., Сабельфельд В. К. Теория схем программ. — М.: Наука, 1991.
6. Летичевский А. А. Функциональная эквивалентность дискретных преобразователей. II // Кибернетика. — 1970. — № 2. — С. 14–28.
7. Ляпунов А. А. О логических схемах программ // Проблемы кибернетики. Вып. 1. — М.: Физматгиз, 1958. — С. 46–74.
8. Ляпунов А. А., Яблонский С. В. Теоретические проблемы кибернетики // Проблемы кибернетики. Вып. 9. — М.: Физматгиз, 1963. — С. 5–22.
9. Нигиян С. А., Хачоян Л. О. О преобразованиях логических программ // Программирование. — 1997. — № 6. — С. 17–28.

10. Подловченко Р. И. Канонические формы схем и их построение в специальных автоматных моделях рекурсивных программ // Программирование. — 1996. — № 5. — С. 3–17.
11. Подловченко Р. И. Система преобразований, полная в классе схем программ с перестановочными операторами // Программирование. — 1998. — № 2. — С. 58–67.
12. Подловченко Р. И. От схем Янова к теории моделей программ // Математические вопросы кибернетики. Вып. 7. — М.: Наука, 1998. — С. 281–302.
13. Подловченко Р. И. Об одном массовом решении проблемы эквивалентных преобразований схем программ // Программирование. — 2000. — № 1. — С. 64–77; № 2. — С. 3–11.
14. Подловченко Р. И., Захаров В. А. Полиномиальный по сложности алгоритм, распознающий коммутативную эквивалентность схем программ // Докл. РАН. — 1998. — Т. 362, № 6. — С. 744–747.
15. Сабельфельд В. К. Эквивалентные преобразования стандартных схем // Проблемы программирования. — Новосибирск, 1976. — С. 94–121.
16. Янов Ю. И. О логических схемах алгоритмов // Проблемы кибернетики. Вып. 1. — М.: Физматгиз, 1958. — С. 75–127.
17. Янов Ю. И. О локальных преобразованиях схем алгоритмов // Проблемы кибернетики. Вып. 20. — М.: Наука, 1968. — С. 201–216.
18. Янов Ю. И. Предельно полная система правил эквивалентных преобразований для программ, вычисляющих всюду определенные функции // Проблемы кибернетики. Вып. 37. — М.: Наука, 1980. — С. 215–238.
19. Manna Z. Mathematical theory of computation. — New York: McGraw Hill Book Co., 1974.
20. Paterson M. S. Program schemata // Machine intelligence. V. 3. — Edinburg: Univ. Press, 1968. — P. 19–31.
21. Rice H. G. Classes of recursively enumerable sets and their decision problems // Trans. Amer. Math. Soc. — 1953. — V. 74, № 2.
22. Sabelfeld V. K. On equivalent transformation of recursion schemes // Theoretical Computer Science and Methods of Computation and Program Construction, INRIA, Grenoble, 1991. — P. 295–312.

Поступило в редакцию 27 VII 2000