УДК 510.5

Г.А. Кравцов, канд. техн. наук, **И.А. Притулюк**, аспирантка
Ин-т проблем моделирования в энергетике им. Г.Е. Пухова НАН Украины
(Украина, 03164, Киев, ул. Генерала Наумова, 15,
тел. (044) 4249165, e-mail: hryhoriy.kravtsov@gmail.com; i.prytulyuk@gmail.com)

Новая классификация алгоритмов

Предложена собственная классификация алгоритмов на основании обзора наиболее известных фундаментальных и современных работ. В отличие от известных классификаций введены понятия алгоритмов высших порядков и контекстнозависимых алгоритмов.

Запропоновано власну класифікацію алгоритмів за результатами огляду найбільш відомих фундаментальних та сучасних робіт. На відміну від відомих класифікацій введено поняття алгоритмів вищих порядків та контекстнозалежних алгоритмів.

К л ю ч е в ы е с л о в а: классификация, свойства, дискретность, детерминированность, случайность, контекстная зависимость.

Изучение теоретических основ функциональных языков программирования приводит к пониманию того факта, что существует несогласованность в теории алгоритмов, а именно между алгоритмами вычислимых функций и понятием функции в функциональных языках.

В работе [1] однозначно определено, что «класс алгоритмов, вычислимых на машинах, в точности совпадает с классом всех частично рекурсивных функций», из чего в последующем сделан вывод о представлении алгоритмов частично рекурсивными функциями. Функциональные языки согласно своей парадигме программирования оперируют вычислимыми на машинах Поста и Тьюринга функциями, класс которых в терминах работы [1] «в точности совпадает с классом всех частично рекурсивных функций», который совпадет с классом алгоритмов, вычислимых на машинах Тьюринга и Поста. К сожалению, в [1] не поясняется, что значит «класс, который в точности совпадает с другим классом», но будем полагать, что в соответствии с теорией классов, построенной на аксиомах Геделя—Бернайса, такое высказывание соответствует понятию равенства.

Широко известные в теории функциональных языков функции высших порядков не имеют аналогов в существующей теории алгоритмов и не выделены в отдельный класс при их классификации. Это стало поводом к

проведению исследования с целью сопоставления двух теорий и устранения несогласованности.

Проведенное исследование основано на детальном изучении получивших наибольшую известность и имеющих практическую ценность работ. Для представления целостной картины будем следовать этим работам, цитируя иногда значительные отрывки оригинальных работ.

Понятие алгоритма. Алгоритмам и их свойствам посвящено большое число фундаментальных работ, среди которых основной можно считать работу [2], в которой Д.Э. Кнут пишет, что само слово «алгоритм» (algorithm) (устаревшее «алгорифм») представляет большой интерес. «На первый взгляд может показаться, что кто-то собирался написать слово «логарифм», но случайно переставил первые четыре буквы. В словаре Webster's New World Dictionary, вышедшем в 1957 г. этого слова еще не было. Там находим только устаревшую форму «algorism» — старинное слово, которое означает «выполнение арифметических действий в помощью арабских цифр». В средние века абакисты считали на абаках (счетных досках), а алгоритмики использовали algorism. В эпоху возрождения это слово оказалось забытым. Одни лингвисты того времени пытались объяснить его значение сочетанием слов «algiros» (больной) и «arithmos» (число), другие не соглашались с таким толкованием и утверждали, что это слово происходит от «King Algor Of Castile».

Наконец, историками математики было установлено истинное происхождение слова «algorism»: оно происходит от имени автора знаменитого персидского учебника по математике Абу Абд Аллах Мухаммед ибн Муса аль-Хорезми, что означает буквально «Отец Абдулы, Муххамед, сын Муссы, уроженец Хорезма». Аль-Хорезми написал знаменитую «Книгу о восстановлении и противопоставлении», которая определила еще одно слово: «алгебра».

Постепенно форма и значение слова «algorism» исказились. Как объясняется в словаре Oxford English Dictionary, это слово «претерпело множество псевдоэтимологических искажений, включая последний вариант «algorithm», где произошла путаница с корнем слова греческого происхождения «arithmetic». Этот переход от «algorism» к «algorithm» кажется вполне закономерным ввиду того, что происхождение рассматриваемого слова было полностью забыто. В старинном немецком математическом словаре Vollständiges mathematisches Lexicon (Leipzig, 1747) дано следующее определение слова algorithmus: «Этот термин включает в себя понятие о четырех типах арифметических операций, а именно: о сложении, умножении, вычитании и делении». Латинское выражение algorithmus infinitesimalis в то время использовалось для определения «способов выполнения действий с бесконечно малыми величинами, открытыми Лейбницем (Leibniz)» [2].

Д.Э. Кнут [2] утверждает, что «современное значение слова «алгоритм» во многом аналогично таким понятиям, как рецепт, процесс, метод, способ, процедура, программа. Но все-таки слово «algorithm» имеет дополнительный смысловой оттенок. Алгоритм — это не просто последовательность выполнения операций для решения задачи определенного типа».

Т. Кормен [3] неформально определяет алгоритм, как «любую корректно определенную вычислительную процедуру, на вход (input) которой подается некая величина или набор величин, и результатом выполнения которой является выходная величина (output) или набор значений. Таким образом, алгоритм представляет собой последовательность вычислительных шагов, преобразующих входные величины в выходные.

Алгоритм также можно рассматривать как инструмент, предназначенный для решения корректно поставленной вычислительной задачи (computational problem). В постановке задачи в общих чертах задаются отношения между входом и выходом. В алгоритме описывается конкретная вычислительная процедура, с помощью которой удается добиться указанных отношений. Говорят, что алгоритм корректен (correct), если для каждого ввода результатом его работы является корректный выход. Мы говорим, что корректный алгоритм решает данную вычислительную задачу. Если алгоритм некорректный, то для некоторых вводов он может вообще не завершить свою работу или выдать ответ, отличный от ожидаемого. Правда, некорректные алгоритмы иногда могут оказаться полезными, если в них есть возможность контролировать частоту возникновения ошибок».

Следует заметить, что что проверка корректности алгоритма может быть весьма трудоемкой задачей. Например, как проверить корректность алгоритма на всем множестве действительных чисел или корректность факторизации произвольного числа N?

З.В. Алферова [4] под алгоритмом понимает «конечную совокупность точно сформулированных правил решения некоторого класса задач. Алгоритмы, в соответствии с которыми решение поставленных задач сводится к арифметическим действиям, называются численными алгоритмами, а алгоритмы, в соответствии с которыми решение поставленной задачи сводится к логическим действиям, называются логическими алгоритмами». В [4] утверждается, что «алгоритмами в современной математике принято называть конструктивно задаваемые соответствия между словами в абстрактных алфавитах, где абстрактным алфавитом называется любая конечная совокупность объектов, называемых буквами или символами данного алфавита. При этом природа этих объектов совершенно не интересует. Символами абстрактных алфавитов можно считать, напри-

мер, буквы алфавита какого-либо языка, цифры, любые значки, рисунки и даже слова конкретного языка. Важно лишь, чтобы рассматриваемый алфавит был конечным.

Можно назвать алфавит конечным множеством различных символов. Слово «абстрактный» для краткости будем опускать. Как любое множество алфавит задается перечислением его элементов, т.е. символов. Под словом или строкой алфавита понимают любую конечную упорядоченную последовательность символов. Наряду со словами положительной длины, состоящими не менее чем из одного символа, в некоторых случаях целесообразно рассматривать также пустые слова, не содержащие ни одного символа. Очевидно, что такое определение слова отличается от принятого в обычном или естественном языке. При изучении алгоритмов словами в указанном выше смысле следует считать любые сочетания и последовательности символов, в том числе и бессмысленные.

Алфавитным оператором или алфавитным отображением называется всякое соответствие, при котором словам некоторого алфавита сопоставляются слова в том же самом или другом фиксированном алфавите. При этом первый алфавит называется входным, второй — выходным алфавитом данного оператора. В случае совпадения входного и выходного алфавитов говорят, что алфавитный оператор задан в соответствующем алфавите. Иными словами, алфавитный оператор — функция, задающая соответствие между словами входного алфавита и словами этого же или другого алфавита.

Различают однозначные и многозначные алфавитные операторы. Под однозначным алфавитным оператором понимается такой алфавитный оператор, который каждому входному слову ставит в соответствие не более одного выходного слова. С математической точки зрения, однозначный алфавитный оператор является сюръективным отображением».

Следуя утверждениям З.В. Алферовой [4], укажем, что алфавитный оператор, не сопоставляющий некоторому входному слову никакого выходного слова, не определен на этом слове. Такие операторы широко используются в функциональных языках программирования в операциях «сопоставления с образцом» (pattern matching) [5] и в частично определенных функциях (partial function) [5, 6].

Совокупность всех слов, на которых алфавитный оператор определен, называется его областью определения или доменом [6]. Совокупность всех слов, которые могут быть результатом выполнения алфавитного оператора, называется областью значений или кодоменом [6].

«С каждым алфавитным оператором связывается интуитивное представление о его сложности. Наиболее простыми являются алфавитные

операторы, осуществляющие посимвольные отображения. Посимвольное отображение состоит в том, что каждый символ входного слова заменяется некоторым символом выходного алфавита. Большое значение имеют так называемые кодирующие отображения. Под кодирующим отображением понимается соответствие, сопоставляющее каждому символу входного алфавита некоторую конечную последовательность символов в выходном алфавите, называемую кодом. Кодирование позволяет сводить изучение произвольных алфавитных отображений к алфавитным отображениям в некотором стандартном алфавите. Наиболее часто в качестве такого стандартного алфавита выбирается так называемый двоичный алфавит, состоящий из двух символов, которые обычно отождествляются с цифрами 0 и 1». Так объясняется природа кодирования в работе [4].

С понятием сопряженного алфавитного оператора, а также с природой взаимосвязи сопряженных алфавитных операторов можно более детально ознакомиться в работе [4]. Следует заметить, что предложенный З.В. Алферовой алфавитный оператор есть не что иное как уточненная абстракция функтора, а сопряженный алфавитный оператор — уточненная абстракция сопряженного функтора в теории категорий [7, 8].

Согласно [4] алфавитные операторы, задаваемые с помощью конечной системы правил, принято называть алгоритмами. Отсюда легко понять, что «всякий алфавитный оператор, который можно фактически задать, является непременно алфавитом. Однако следует иметь в виду одно различие, существующее между понятиями алфавитного оператора и алгоритма. В понятии алфавитного оператора существенно лишь само соответствие, устанавливаемое между входными и выходными словами, а не способ, которым это соответствие устанавливается. В понятии алгоритма, наоборот, основным является способ задания соответствия, устанавливаемого алгоритмом». Таким образом, алгоритм — это алфавитный оператор вместе с правилами, определяющими его действия.

В работе [9] дано определение алгоритмов, как «некоторых инструментов оперирования структурами данных, где структурой данных называется способ организации данных в памяти компьютера (или на диске). Примерами структур данных являются массивы, связанные списки, стеки, двоичные деревья, хеш-таблицы и т.п. Очевидно, что такое определение малоинформативное».

А.И. Мальцев [1] констатирует интуитивность понятия алгоритма, которое «хотя и нестрогое, но настолько ясное, что практически не было серьезных случаев, когда математики разошлись бы во мнениях относительно того, является ли алгоритмом тот или иной конкретно заданный процесс». Однако, «...положение существенно изменилось в XX веке, выд-

винувшем на первый план такие алгоритмические проблемы, существование положительного решения которых было сомнительным. Действительно, одно дело — доказать существование алгоритма, другое — доказать отсутствие алгоритма. Первое можно сделать путем фактического описания процесса, решающего задачу; в этом случае достаточно и интуитивного понятия алгоритма, чтобы удостовериться в том, что описанный процесс и есть алгоритм. Доказать несуществование алгоритма таким путем невозможно. Для этого нужно точно знать, что такое алгоритм.

В 20-х годах XX века задача точного определения понятия алгоритма стала одной из центральных математических проблем. Решение ее было получено в середине 30-х годов того же века в работах Гильберта, Геделя, Черча, Клини, Поста и Тьюринга в двух форматах. Первое решение было основано на понятии рекурсивной функции, второе — на описании точно очерченного класса процессов».

Согласно [1] «числовые функции, значения которых можно вычислять посредством некоторого (единого для данной функции) алгоритма, называются вычислимыми функциями. Именно определение А.И. Мальцева дает возможность построить параллель между функциями в функциональных языках программирования и теоретическими аспектами алгоритмов в математике, ибо совокупность вычислимых функций для самых разных пониманий процессов, удовлетворяющая условиям, определенным Мальцевым для алгоритма, оказалась легко описываемой в терминах математики. Эта точно описанная совокупность числовых функций, совпадающая с совокупностью всех вычислимых функций при самом широком до сих пор известном понимании алгоритма, носит название совокупности рекурсивных функций».

В работе [1] сказано: «Гильберт и Бернайс [10] рассматривали рекурсивные определения, отталкиваясь от аксиом арифметики Пеано и выстраивая свое видение рекурсивной арифметики. Опираясь на идеи Гильберта и Бернайса, Гедель [11] впервые описал класс всех рекурсивных функций как класс всех числовых функций, определимых в некоторой формальной системе. Исходя из других соображений, Черч [12] пришел к тому же классу числовых функций, что и Гедель. Анализ идей, приведших к этому классу функций, позволил Алонзо Черчу первым опубликовать гипотезу о том, что класс рекурсивных функций тождественен классу всюду определенных вычислимых функций. Эта гипотеза известна как тезис Черча. Так как понятие вычислимой функции точно не определено, то тезис Черча доказать нельзя.

Учитывая, что не все процессы вычислений конечны, Стив Колл Клини ввел понятие частично рекурсивной функции [13] и высказал гипотезу,

что все частичные (т.е. не обязательно определенные для всех значений аргументов) функции, вычислимые посредством алгоритмов, являются частично рекурсивными».

В работе [1] не различаются гипотезы А. Черча и С. Клини. Гипотеза С. Клини, которая является более общей, чем гипотеза А. Черча, так же недоказуема, как и гипотеза А. Черча. «Тезис Черча оказался достаточным, чтобы придать необходимую точность формулировкам алгоритмических проблем и в ряде случаев сделать возможным доказательство их неразрешимости» [1]. А.И. Мальцев констатирует, что «точное описание класса частично рекурсивных функций вместе с тезисом Черча — Клини дает одно из возможных решений задачи об уточнении понятия алгоритма».

Э.Л. Пост [14] и А. Тьюринг [15] «независимо друг от друга и почти одновременно с работами Черча и Клини уточнили непосредственно само понятие алгоритма и уже потом, при его помощи точно определили класс вычислимых функций. Основная мысль Поста и Тьюринга заключалась в том, что алгоритмические процессы — это процессы, которые может совершать подходяще устроенная «машина». В соответствии с этой мыслью ими были описаны в точных математических терминах узкие классы машин, позволяющих осуществить или имитировать очень сложные алгоритмические процессы, которые когда-либо описывались математиками» [1].

Свойства алгоритма. Согласно [2] алгоритм обладает такими пятью свойствами: конечность, определенность, точка входа (ввод), точка выхода (вывод) и эффективность.

Конечность алгоритма означает, что алгоритм всегда должен заканчиваться после выполнения конечного числа шагов. Число шагов алгоритма может быть достаточно большим, но целесообразным. Согласно [4] свойство алгоритма, обеспечивающее получение результата через конечное число шагов, называется результативностью алгоритма. Однако автор [2], дав определение свойства конечности алгоритма, не сказал ни слова о бесконечных алгоритмах. Под бесконечным алгоритмом будем понимать такой алгоритм, реализация которого в виде программы при запуске на компьютере может быть остановлена только посредством выключения (перезагрузки) компьютера.

Следует заметить, что «процедура, обладающая всеми характеристиками (свойствами) алгоритма, за исключением, возможно, конечности, называется методом вычислений. Метод вычисления, выраженный на языке программирования, называется программой» [2].

Определенность [2] «требует, чтобы каждый шаг алгоритма был точно определен. Действия, которые нужно выполнить, должны быть строго и недвусмысленно определены для каждого возможного случая. При опи-

сании алгоритмов естественными языками существует возможность неточного понимания мысли автора. Чтобы преодолеть это затруднение, для описания алгоритмов были разработаны формально определенные языки — языки программирования или машинные языки, в которых каждый оператор имеет строго определенное значение.

Каждый алгоритм имеет точку входа или ввода. В точке входа определяются входные данные алгоритма, которые могут и отсутствовать. Входные данные могут быть статическими, а могут быть динамическими, которые берутся из определенного набора объектов. Входные данные можно поделить на две группы: определяющие алгоритм и контекстные. Определяющие алгоритм данные представляют собой необходимый и достаточный набор начальных данных для решения некоторой задачи, описываемой алгоритмом в ее теоретической постановке. Контекстные данные — данные, определяющие условия проведения вычислений».

Например, необходимо отсортировать последовательность чисел в один поток или несколько потоков вычислений. Суть задачи от этого не меняется: необходимо отсортировать последовательность. Исходная последовательность — это определяющие алгоритм исходные данные, а параметр «в сколько потоков сортировать» — это контекстные исходные данные. Поскольку алгоритм есть функция, контекстно-зависимые алгоритмы порождают класс функций в функциональных языках программирования, например Scala, называемый замыканием (closure) [5].

Каждый алгоритм имеет вывод, состоящий из одного или нескольких выходных данных (результатов работы алгоритма). Результаты работы алгоритма тесно связаны с входными данными.

Алгоритм обычно считается эффективным, если все его операторы достаточно просты для того, чтобы их можно было точно выполнить в течение конечного промежутка времени с помощью карандаша и бумаги. Понятие эффективности алгоритма связано с появлением научного направления — анализ алгоритмов [1, 3]. Понятие эффективность алгоритма применимо только к алгоритмам чистых вычислений. Выполнение каждого из операторов алгоритма не означает, что с его помощью можно получить результат за приемлемое время для произвольной задачи из класса решаемых им. В первую очередь, это задачи из класса *NP*-сложных или экспоненциальной сложности. Данное замечание требует отдельного исследования.

К свойствам алгоритма относят детерминированность, массовость, эквивалентность, область применимости [2, 4]. З.В. Алферова определяет: «Два алфавитных оператора считаются равными, если они имеют одну и ту же область определения и сопоставляют любому наперед заданному

входному слову из этой области одинаковые выходные слова» [4]. Д. Кнут утверждает, что «два алгоритма считаются равными, если равны соответствующие им алфавитные операторы и совпадает система правил, задающих действие этих алгоритмов на выходные слова» [2].

Согласно З.В. Алферовой: «Эквивалентность алгоритмов определяется через совпадение алфавитных операторов алгоритмов, но не совпадение способов их задания (системы правил). Обычно в теории алгоритмов рассматривают лишь такие алгоритмы, которым соответствуют однозначные алфавитные операторы. Всякий алгоритм такого рода любому входному слову относит только одно выходное слово. Такие алгоритмы и алфавитные операторы называют детерминированными.

Массовость алгоритма — свойство алгоритма быть применимым для множества строк. Если алгоритм применим для множества строк, то он обладает свойством массовости.

Из свойства результативности (конечности) вытекает понятие области применимости алгоритма.

Область применимости алгоритма — множество строк, для которых алгоритм конечен или результативен.

Теперь эквивалентность алгоритмов может быть определена следующим образом: два алгоритма эквивалентны, если совпадают их области применимости и результаты переработки любого слова из этой области.

Различают еще случайные и самоизменяющиеся алгоритмы. Алгоритм называется случайным, если в системе правил, описывающих алгоритм, предусматривается возможность случайного выбора тех или иных слов или тех или иных правил. Им соответствуют многозначные алфавитные операторы». Субъективно понятие применимости требует уточнения, а понятие результативности вообще не определено.

«Алгоритм называется самоизменяющимся, если он не только перерабатывает входные слова, но и сам изменяется в процессе такой переработки. Результат действия самоизменяющегося алгоритма на то или иное слово зависит не только от этого слова, но и от истории предыдущей работы алгоритма» [4].

Изучение теории функциональных языков [6, 12] и их практического применения приводит к необходимости формулирования понятия алгоритма чистых вычислений или чистого алгоритма.

Чистый алгоритм — алгоритм решения конкретной задачи, в котором не существует ни одного шага, который может быть удален без нарушения отображения исходных данных в выходные. Поясним определение. Например, есть алгоритм, который вычисляет некоторое значение в цикле и публикует промежуточные результаты. Шаги публикации

промежуточных данных могут быть удалены из алгоритма без ущерба для конечного результата вычислений. Чистые алгоритмы в функциональных языках программирования (например, Scala) представляют в виде чистых функций (pure function) [5]. Шаги алгоритмов, которые могут быть удалены без ущерба для вычислений, принято называть сторонний эффект (side effect)[16].

В теории алгоритмов большое внимание уделяется общим способам задания алгоритмов, для которых характерны свойства универсальности, т.е. таким способам, которые позволяют задать алгоритм, эквивалентный любому наперед заданному алгоритму. Всякий общий способ задания алгоритмов принято называть алгоритмической системой. При описании алгоритмических систем используются специальные формализованные средства, в том числе формальная семантика.

Основные формализмы прикладной теории алгоритмов можно разделить на два направления, условно называемые «алгебраическим» и «геометрическим» [1]. Алгебраическая теория строится в конкретной символике, при которой алгоритмы рассматриваются в виде линейных текстов. В геометрической теории алгоритмы строятся в виде множеств, между которыми вводятся связи, носящие характер отображений или бинарных отношений. При этом значительное место занимает геометрическая интерпретация объектов в виде графов, вершины которых задают элементы множества, а ребра — отношения между ними.

К первому направлению относятся рекурсивные функции, машины Тюринга, операторные системы Ван-Хао, А.А. Ляпунова, логические схемы алгоритмов Ю.И. Янова и др.

Ко второму направлению относятся представления нормальных алгоритмов А.А. Маркова [17] (применимых к словам в различных алфавитах), граф-схем, блок-схемный метод алгоритмизации и др.

Наиболее удобными при коллективной работе являются алгебраические представления. Поэтому теория категорий [8] играет важную роль. Композиция как основной механизм изучения категорий позволяет представлять чистые алгоритмы в виде математических категорий.

Алгоритм высшего порядка — это алгоритм, принимающий в качестве входного параметра другой алгоритм. Аналогами таких алгоритмов в функциональных языках программирования, например Scala, являются функции высших порядков (higher-ordered function, HOF)[15, 16]. Следует заметить, что алгоритмом первого порядка является алгоритм, не принимающий в качестве параметра другой алгоритм.

В работе [18] дано интуитивное определение алгоритма и сделан акцент на том, что алгоритмы следует всегда рассматривать в совокуп-

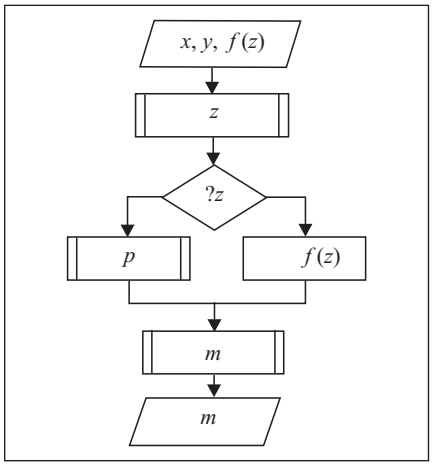
ности со структурами данных. «Алгоритм можно определить, описав процедуру решения задачи на естественном языке или в виде компьютерной программы, которая реализует процедуру», «термин алгоритм применятся в вычислительной технике для описания конечного, детерминированного и эффективного метода решения задачи, который годится для реализации в виде компьютерной программы». Из этих интуитивных определений можно сделать вывод о том, что Р. Седжвик и К. Уэйн [18] выделяют следующие свойства алгоритма: конечность, детерминированность, эффективность и массовость.

Свою систему свойств алгоритмов формирует А.И.Мальцев [1]: «Алгоритм — это процесс последовательного построения величин, идущий в дискретном времени таким образом, что в начальный момент задается исходная конечная система величин, а в каждый следующий момент система величин получается по определенному закону (программе) из системы величин, имевшихся в предыдущий момент времени (дискретность алгоритма). При этом система величин, получаемых в какой-то (не начальный) момент времени, однозначно определяется системой величин, полученных в предыдущие моменты времени (детерминированность алгоритма)».

Закон получения последующей системы величин из предшествующей должен быть простым и локальным, что определяет еще одно свойство алгоритма по Мальцеву — элементарность шагов алгоритма. Однако автор [1] не поясняет, что он подразумевает, говоря о «простом и локальном», оставляя это на суд читателя.

Следующее утверждение характеризует направленность алгоритма: «Если способ получения последующей величины из какой-нибудь заданной величины не дает результата, то должно быть указанно, что надо считать результатом алгоритма» [1].

Алгоритмы высших порядков. Авторы работы [18] придерживаются мнения, что «структуры данных существуют как побочные или конечные продукты алгоритмов, и их изучение необходимо для понимания алгоритмов». Они также говорят о том, что следует выделить в отдельный класс алгоритмы первого порядка и алгоритмы высших порядков. Это вытекает из следующего: «Простые алгоритмы могут потребовать сложные структуры данных, и наоборот, сложные алгоритмы могут использовать простые структуры данных» [18]. Очень важно понимать, что если некоторый алгоритм требует всегда одну и ту же структуру данных, то такой алгоритм не является алгоритмом высшего порядка. В таком случае следует говорить о композиции алгоритмов или композиции морфизмов или функторов, если речь идет об алгоритмах вычислимых функций или чистых алгоритмах [8].



Пример алгоритма высшего порядка

При рассмотрении вопроса о выделении порядка алгоритмов как отдельного классификационного признака наиболее часто возникает вопрос: являются ли алгоритмы высших порядков суперпозицией алгоритмов?

Согласно [19] «неформальное пошаговое определение алгоритма показывает четыре способа соединения алгоритмов:

1. Последовательное соединение, или суперпозиция, когда за выполнением шагов первого алгоритма следует выполнение шагов второго алгоритма и входной информацией для второго алгоритма служит выходная информация первого алгоритма.

2. Композиция, когда два алгоритма параллельно и независимо выполняют работу над входной информацией и результатом их работы является выходная информация обоих алгоритмов.

3. Ветвление, когда в зависимости от выполнения условия для входной информации, проверяемого первым алгоритмом, выполняется или не выполняется второй алгоритм.

4. Цикл, когда при проверке первым алгоритмом выполнения условия для входной информации соединения алгоритмов или выходной информации предыдущего выполнения второго алгоритма этот второй алгоритм снова выполняется в цикле или происходит выход из цикла с выходной информацией второго алгоритма».

Рассмотрим на примере заявленные в работе [19] способы соединения.

На рисунке представлен алгоритм, принимающий на вход параметры x, y и $f(z)$, где $f(z)$ — алгоритм от параметра z; $?z$ — некоторое условие бинарного ветвления; прямоугольниками с двойными линиями обозначены предопределенные шаги алгоритма. Прямоугольником, выполняющим преобразование $f(z)$, обозначена реконфигурируемая часть алгоритма. Очевидно, что в данном примере $f(z)$ находится в суперпозиции по отношению к z, а m — в суперпозиции по отношению к $f(z)$. Однако невозможно показать, что изображенный алгоритм находится в суперпозиции по отношению к $f(z)$, так же как и не является композицией, ветвлением или циклом.

Новая классификация. Изложенное позволяет представить свойства алгоритма в виде таблицы, в которой на пересечении строк и столбцов знаком «+» отмечено свойство, изучаемое определенным автором. Как видно из таб-

Свойство алгоритма	А.И. Мальцев [1]	Т.Х. Кормен [3]	З.В. Алферова [4]	Д.Э. Кнут [2]	Р. Седжвик [18]
Дискретность	+				
Детерминированность	+		+		+
Элементарность шагов	+				
Самоизменяемость			+		
Чистота	+				
Порядок	Ранее не определялось				
Эффективность		+		+	+
Массовость	+				+
Конечность	+	+			+
Эквивалентность			+		
Направленность	+				
Применимость			+		
Корректность		+			
Контекстная зависимость	Ранее не определялось				

лицы, любой алгоритм, обладающий свойствами дискретности, элементарности шагов, эффективности, направленности, массовости и эквивалентности, может быть дополнительно определен как детерминированный либо стохастический; самоизменяемый либо несамоизменяемый; первого либо высшего порядка; контекстнозависимый либо контекстносвободный.

Четыре определенных критерия деления алгоритмов на классы формируют новую классификацию алгоритмов.

Выводы

Предложенная классификация алгоритмов, основанная на анализе наиболее известных фундаментальных и современных работ, позволяет ввести понятие алгоритмов высших порядков и контекстнозависимых алгоритмов, проводя четкую параллель между алгоритмами и их представлением в функциональных языках программирования, восстанавливая согласованность двух теорий.

СПИСОК ЛИТЕРАТУРЫ

1. *Мальцев А.И.* Алгоритмы и рекурсивные функции. — М.: «Наука», 1986. — 367 с.
2. *Кнут Д.Э.* Искусство программирования. Т. 1. Основные алгоритмы / Пер.с агл.: уч. пос., 3-е изд., под общей ред. докт. физ.-мат. наук, проф. Ю.В. Козаченко. — М. : Изд. дом «Вильямс», 2000. — 720 с.
3. *Кормен Т.Х., Лейзерсон Ч.И., Ривест Р.Л., Штайн К.* Алгоритмы: построение и анализ. 2-е изд. — М.: «Вильямс», 2005. — 1290 с.

4. *Алферова З.В.* Теория алгоритмов. — М. : Статистика, 1973. — 164 с.
5. *Subramaniam V.* Programming Scala. Tackle Multicore Complexity on Java Virtual Machine. The Progmatic Programmers. — Pragmatic Bookshelf. — Raleigh, North Carolina, Dallas, Texas, 2008. — 218 p.
6. *Пирс Б.* Типы в языках программирования / Пер. с англ. —М.: «Лямбда пресс» & «Добросвет», 2012. — 655 с.
7. *Foundational* Theories of Classical and Constructive Mathematics. Ed. Giovanni Sommaruga. — Springer: The Western Ontario Series in Philosophy of Science, 2011. — 316 p.
8. *Маклейн С.* Категории для работающего математика.—М.: ФИЗМАТЛИТ, 2004.— 352 с.
9. *Лафоре Р.* Структуры данных и алгоритмы в Java // Классика Computer Science. — М. : Изд-во «Питер», 2013. — 704 с.
10. *Гильберт Д., Бернайс П.* Основания математики. Логические исчиления и формализация арифметики. — М. : «Наука», 1982. — 550 с.
11. *Gödel K.* Uber formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I //Monatshefte für mathematik und physik. — 1931. — Т. 38, № 1. — С. 173—198.
12. *Барендрегт Х.П.* Ламбда-исчисление. Его синтаксис и семантика. — М. : Мир, 1985. — 606 с.
13. *Kleene S.C.* General recursive functions of natural numberse // Mathematische Annalen.— 1936. — **112**. — P. 727—742.
14. *Успенский В.А.* Машина Поста. — М.: Наука, 1979. — С. 89—95.
15. *Turing A.* On computable numbers, with an application to the Entscheidungs problem // Proc. of the London Mathematical Society. Series 2. — 1936-7. — **42**. — P. 230—265. — Last access: August of 2015. — Mode of access: http://www.cs.virginia.edu/~robins/Turing_Paper_ 1936.pdf .
16. *Хорстман К.* Scala для нетерпеливых /Пер. с англ. — М.: Изд-во «ДМК», 2013. — 407 с.
17. *Марков А.А.* Теория алгорифмов // Тр. Математич. ин-та им. В.А. Стеклова. Т. 42. — М.: МАИК «Наука/Интерпериодика», 1954. — 376 с.
18. *Седжвик Р., Уэйн К.* Алгоритмы на Java, 4-е изд. — М. : «И.Д. Вильямс», 2013. — 848 с.
19. *Рублев В.С.* Основы теории алгоримов.— Ярославль: Ярославский гос. ун-т им. П.Г. Демидова, 2005. — 143 с.

H.A. Kravtsov, I.A. Prytulyuk

A NEW ALGORITHM CLASSIFICATION

The author's classification of algorithms based on the review of famous fundamental and modern works is presented. The author's classification is different from already known ones due to the involved terms of high order algorithms and context-related algorithms.

K e y w o r d s: classification, properties, discreteness, determinism, probability, context dependency.

REFERENCES

1. Maltsev, A. (1986), *Algoritmy i rekursivnyie funktsii* [Algorithms and recursive functions], Nauka, Moscow, Russia.
2. Knut, D. (1968), *Iskustvo programmirovaniya. Tom1. Osnovnyie algoritmy* [The Art of Computer Programming. Vol. 1. Fundamental Algorithms], Manual, Third Edition, translated from English by Prof. Kozachenko, Yu.V., Williams, Moscow, Russia.
3. Cormen, T.H., Leizerson, Ch.I., Rivest, R.L. and Shtain, K. (1990), *Algoritmy: postroyeniye i analiz* [Algorithms: structure and analysis], Williams, Moscow, Russia.

4. Alferova, Z.V. (1973), *Teoriya algoritmov* [Algorithm theory], Statistika, Moscow, Russia.
5. Subramaniam, V. (2008), Programming Scala. Tackle multicore complexity on Java virtual machine. The pragmatic programmers, Pragmatic bookshelf, Raleigh, North Carolina, Dallas, Texas, USA.
6. Pirs, B. (2012), *Tipy v yazykakh programmirovaniya* [Types in programming languages], Lambda Press and Dobrosvet, Moscow, Russia.
7. Foundational theories of classical and constructive mathematics (2011), Ed. by Giovanni Sommaruga, Springer: The Western Ontario Series in Philosophy of Science, Canada.
8. Makleyn, S. (1998), *Kategorii dlya rabotayushchego matematika* [Categories for the working mathematician], Translated from English, Fizmatlit, Moscow, Russia.
9. Lafore, R. (2002), *Struktury dannykh i algoritmy v Java* [Data structures and algorithms in Java], Piter, Moscow, Russia.
10. Gilbert, D. and Bernays, P. (1982), *Osnovaniya matematiki. Logicheskie ischileniya i formalizatsiya arifmetiki* [Mathematics bases. Logical calculus and arithmetic formalization], Nauka, Moscow, Russia.
11. Gödel, K. (1931), Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I, *Monatshefte für mathematik und physic*, Vol. 38, no. 1, pp. 173-198.
12. Barendregt, H.P. (1985), *Lambda-ischislenie. Yego sintaksis i semantika* [The Lambda calculus. Its syntax and semantics], Translated from English, Mir, Moscow, Russia.
13. Kleene, S.C. (1936), General recursive functions of natural numbers, *Mathematische Annalen*, Vol. 112, pp. 727-742.
14. Uspenskiy, V. (1979), *Mashina Posta* [Post–Turing machine], Nauka, Moscow, Russia.
15. Turing, A. (1936), On computable numbers, with an application to the Entscheidungs problem, *Proc. of the London Mathematical Society,* Series 2, Vol. 42, pp. 230-265, available at: http://www.cs.virginia.edu/~robins/Turing_Paper_1936.pdf (accessed August 2015).
16. Khorstman, C. (2012), *Skala dlya neterpelivykh* [Scala for the impatient], translated from English, DMK, Moscow, Russia.
17. Markov, A. (1954), "Algorithm theory", *Trudyi matematich. in-ta im. V. A. Steklova,* Vol. 42, MAIK «Nauka/Interperiodika», Moscow, Russia.
18. Sedzhvik, R. and Ueyn, K. (2011), *Algoritmy na Java* [Algorithms in Java], Williams, Moscow, Russia.
19. Rublev, V. (2005), *Osnovyi teorii algorimov* [Fundamentals of algorithm theory], P.G. Demidov Yaroslavl State University, Yaroslavl, Russia.

Поступила 23.02.16

КРАВЦОВ Григорий Алексеевич, канд. техн. наук, докторант Ин-та проблем моделирования в энергетике им. Г.Е. Пухова НАН Украины. В 2000 г. окончил Севастопольский военно-морской ин-т им. П.С. Нахимова. Область научных исследований — кибербезопасность смарт-грид, криптография, программирование, разработка распределенных гетерогенных вычислительных систем.

ПРИТУЛЮК Ирина Афанасьевна, аспирантка Ин-та проблем моделирования в энергетике им. Г.Е. Пухова НАН Украины. В 2011 г. окончила Открытый международный университет развития человека «Украина». Область научных исследований — моделирование финансовых систем, теория игр.