Косовский Николай Кириллович

УДК 510.52

АЛГОРИТМЫ МАРКОВА-ТУРЧИНА И ДОКАЗАТЕЛЬСТВА ПОЛИНОМИАЛЬНОЙ ЭФФЕКТИВНОСТИ ПРОГРАММ НА ЯЗЫКЕ РЕФАЛ-5

Аннотация

Вводятся алгоритмы Маркова-Турчина как обобщение алгоритмов, введённых автором в [5], и являющиеся существенной модификацией программ, написанных на языке рефал-5. Предлагается критерий их эффективности для реализации на машине Тьюринга за полиномиальное число шагов. Этот критерий не использует ограничения памяти, в отличие от того, как это было установлено в [3] для дважды полиномиальных программ на языке рефал-5. Критерий основан на специального вида рекурсии в программах, являющейся существенным обобщением хвостовой рекурсии, предложенной Т. Борландом для программ на турбо прологе. Это обобщение связано с основным типом данных в языке рефал-5, представляющем собой древовидно структурированные тексты посредством круглых скобок. Для алгоритмов Маркова-Турчина (со встроенным динамическим интерпретатором) просто доказываются некоторые модификации теорем теории сложности алгоритмов, полезные для математиков-программистов. Иначе говоря, вводится математическое понятие алгоритма, модифицирующее (в основном, упрощающее, а также объединяющее в единый алгоритм все вспомогательные алгоритмы с помощью использования встроенного интерпретатора) понятие рефал-5 функции, введённой В.Ф. Турчиным (см., например, [2, 6]), в частности, путем расширения понятия нормального алгоритма А.А. Маркова на древовидно оформленные тексты с помощью структурных (круглых) скобок. Введение встроенного динамического интерпретатора (в терминологии В.Ф. Турчина – расширения встроенной метауниверсальной функции), вычисляющего применение любого алгоритма с динамически полученной его записью к исходным данным, является существенным расширением понятия алгоритма Маркова. Известно, что на практике алгоритмы и исходные данные ограничены по длине. В статье также доказываются теоремы о вариантах проблемы применимости алгоритмов Маркова-Поста с учётом этого обстоятельства. Точнее, доказываются нижние оценки длины программ, разрешающих проблему применимости коротких программ к коротким данным на языке алгоритмов Маркова-Турчина (со встроенным динамическим интерпретатором).

Ключевые слова: машина Тьюринга, полиномиальное число шагов, рефал-5, нормальный алгоритм Маркова, класс сложности **FP**, проблема применимости алгоритмов к данным.

ВВЕДЕНИЕ

Исходный признак эффективности для программ, написанных на рефале, был предложен В.Ф. Турчиным. Он состоит в отсутствии дублирования в правых частях правил (то есть после знака равенства) переменных (как для выражений, так и для термов языка рефал). Этот признак был уточнён автором в [3]. Ниже предлагается признак полиномиальной эффективности рефал-5 функции, основанный на верхней оценке длины аргумента её рекурсивного вызова и на отсутствии её рекурсивных вызовов в аргументах каких-либо функций.

Известно, что рекурсия в виде итерации эффективно реализуется на компьютере. Для эффективности алгоритмов на языке турбо пролог Т. Борландом [8] была предложена хвостовая рекурсия. Ниже она обобщается под названием сборочная рекурсия и используется для характеризации рефал-5 функций, вычислимых за полиномиальное число шагов на машине Тьюринга.

Первоначальные модели вычислений, приближенные к компьютерам, имели, как правило, линейный характер памяти: машины Тьюринга, нормальные алгоритмы Маркова, РАМ, РАСП [1]. Теория сложности алгоритмов, как правило, использует эти алгоритмы [7] или алгоритмы Маркова-Поста [3]. В то же время, программирование обычно использует не только линейное представление данных, но и древовидное представление текстов, удобно представимое, например, в языках серии рефал, разработанных В.Ф. Турчиным (см. например, [2, 6]). В статьях [3, 4] исследованы варианты алгоритмов Маркова-Поста с точки зрения способов верхней оценки полиномиальной по времени сложности.

В настоящей статье, в частности, для целей обучения одному из разделов математической информатики используется эффективно реализуемое математическое понятие алгоритма, предназначенное для обработки древовидно упорядоченных (с помощью круглых скобок) текстов на основе правил языка рефал-5 с дополнительным включением встроенного динамического интерпретатора этого языка. С помощью него коротко записывается универсальный алгоритм, играющий важную роль в теории алгоритмов. Вводится понятие алгоритма Маркова-Турчина для обучения программистов способам эффективного программирования алгоритмов переработки древовидно структурированных текстов.

С помощью таких понятий алгоритмов удобно излагать и доказывать теоремы, формулируемые для коротких программ и коротких исходных данных.

Изучение математических понятий алгоритмов в следующей последовательности (машина Тьюринга, алгоритм Маркова, алгоритм Маркова-Поста, сборочный алгоритм Маркова-Турчина, сборочный алгоритм Маркова-Турчина с охраняемыми правилами и, наконец, сборочный алгоритм Маркова-Турчина с древовидными правилами, а также алгоритм Маркова-Турчина) может завершаться изучением языка рефал-5, в котором рекурсия обычно используется существенно не итеративным способом.

Использование компьютеров предполагает, что программы, исходные данные для них, использованные программой память и число её шагов являются всегда ограниченными сверху, например, по длине их записи.

Однако в теории сложности алгоритмов понятия верхних оценок и памяти, и числа шагов определяются с точностью до конечного числа исключений (см. например [7]). Поэтому редкие результаты теории сложности алгоритмов могут представлять существенный интерес для практической математической информатики. Но именно такие результаты, оформленные в виде доказанных теорем, представлены здесь.

Некоторые доказываемые в статье теоремы по существу являются уточнениями для коротких алгоритмов Маркова-Турчина широко известных в теории алгоритмов теорем, например теоремы об алгоритмической неразрешимости проблемы применимости универсального алгоритма (следствие 1 теоремы 2).

1. ИСХОДНЫЕ ОПИСАНИЯ И УТВЕРЖДЕНИЯ

Определение. *Рекурсию назовём сборочной, если все рекурсивные правила, описывающие функцию, являются сборочными, то есть каждый рекурсивный вызов в этих правилах не является частью аргумента какой-либо функции.*

Традиционная реализация сборочных правил на компьютере возможна с помощью смешанных вычислений. Экономичность реализации этих правил связана с тем, что можно подставлять, а не хранить (как правило, в стеке) каждое значение каждой переменной (как для символов и выражений, так и для термов) на каждом шаге рекурсивного вызова основной определяемой функции.

Определение. *Функция принадлежит классу* **FP** *тогда и только тогда, когда она может быть вычислена на машине Тьюринга в рамках полиномиального от длины исходных данных числа шагов.*

Определение. *Функция языка рефал-5 называется бесстековой, если она не содержит встроенных операций над встроенным стеком языка рефал-5.*

В языке рефал-5 возможна запись вложенных условий, что может породить дерево условий. Последовательность условий с предшествующим выражением, выполнение которых записывается вычислением одного из заключительных функциональных выражений, назовём веткой условий для него.

Лемма 1. *Пусть*

1) сборочная бесстековая рефал-5 функция при своём вычислении выполняет полиномиально ограниченное сверху максимальное число выполнения своих рекурсивных вызовов, измеряемое относительно длины исходного аргумента;

2) каждая ветка условий для каждого правила, определяющего эту функцию (включая функциональное выражение, на которое, согласно этой ветке, необходимо заменить исходный образец) содержит не более одного её рекурсивного вызова;

3) длина каждого постоянного (полученного в процессе вычисления) аргумента (находящегося сразу после знака равенства до разделителя – точки с запятой или правой фигурной скобки) рекурсивного вызова этой функции не превосходит длины исходного аргумента правила основной определяемой функции, сложенной с константой, одной и той же для всех применений этого правила;

4) все вспомогательные функции принадлежат классу **FP** *(в частности, метауниверсальная функция применяется только к аргументам, начинающимся с имён функций, принадлежащих* **FP**).

Тогда длина использованной памяти этой функции не превосходит полинома от длины исходного аргумента основной определяемой функции.

Доказательство. Для каждого выполненного рекурсивного вызова основной функции длина постоянного полученного в процессе вычисления аргумента ограничена сверху полиномом от длины исходного аргумента, так как при каждом рекурсивном вызове длина его постоянного аргумента увеличивается не более чем на одну и ту же константу и выполнено условие 1).

Длина результата вычисления правой части каждого правила основной функции в результате применения каждого шага этого правила ограничена сверху полиномом от длины аргумента этого правила. Но сумма полиномиального числа слагаемых, являющихся полиномами, также является полиномом. Поэтому верхняя граница длины используемой памяти основной функции является полиномом. Лемма доказана.

Отметим, что если используется условие рекурсивного спуска (то есть уменьшение длины постоянного аргумента при каждом рекурсивном вызове) вместо условия 2) в лемме 1 и исключается её условие 1), то в этом случае можно вычислить функцию, не принадлежащую классу FP. Её определение может базироваться на следующей рекурсии

$$g(\Lambda) = 1,\ g(e_0 s_1) = g(e_0)g(e_0),$$

где e_0 – слово, а s_1 – символ. Здесь и далее

Λ используется для обозначения пустого слова.

Действительно, можно доказать, что значение функции g от слова длины n равно 1...1, где 1 повторена 2^n раз, а число выполняемых рекурсивных вызовов равно $2^{n+1} - 2$.

Если использовать вспомогательную функцию, удваивающую запись аргумента, то, использовав её в рекурсивном равенстве предыдущей рекурсии, устанавливаем необходимость использования в лемме 1 условия сборочности исходной функции.

Если функция использует встроенный стек языка рефал-5, то можно построить экспоненциального вида функцию, осуществляющую на каждом рекурсивном шаге возведение в квадрат элемента (большего единицы) стека. Этот пример подтверждает необходимость использования в лемме 1 бесстековых функций.

Понятие выражения определяется рекурсивно. В этом определении под алфавитом будем понимать алфавит печатаемых символов (включая пробел) на клавиатуре компьютера. Термином <слово> будем обозначать непустое слово в этом алфавите.

Определение выражения.

- Пустое слово является выражением;
- ′<слово>′ является выражением;
- $s_{\text{<буква>}}, s_{\text{<цифра>}}$ являются выражениями (переменными для символов);
- $e_{\text{<буква>}}, e_{\text{<цифра>}}$ являются выражениями (переменными для постоянных выражений);
- $t_{\text{<буква>}}, t_{\text{<цифра>}}$ являются выражениями (переменными для термов, то есть для символов или выражений, записанных в специальные структурные круглые скобки);
- (<выражение>) является выражением;
- <выражение><выражение> является выражением.

Постоянным выражением назовём выражение, не содержащее переменных. Пусть f – запись алгоритма.

Вызов алгоритма f над выражением e будем обозначать $\langle fe\rangle$. По существу, это обозначение соответствует использованию введенной В.Ф. Турчиным встроенной метауниверсальной функции, вычисляющей применение любого алгоритма (с динамически построенной его записью) к указанным данным, обозначенным посредством e.

Рекурсивный вызов над выражением e будем обозначать $[e]$.

Определение функционального выражения.

- <выражение> является функциональным выражением;
- $\langle f$<выражение>$\rangle$ является функциональным выражением;
- [<выражение>] является функциональным выражением.

Определение. *Правила простейшего алгоритма Маркова-Турчина имеют ровно один из следующих видов:*

- <выражение> $\rightarrow\cdot$ <функциональное выражение>,
- <выражение> $\rightarrow$ [<функциональное выражение>].

При этом все переменные из функциональных выражений встречаются в соответствующем выражении правила, находящемся до знака $\rightarrow$.

Выражение, находящееся до знака $\rightarrow$, будем называть образцом для сравнения.

Определение. *Последовательность правил, разделённых точкой с запятой, заключённая в фигурные скобки, называется простейшим алгоритмом Маркова-Турчина.*

Применению правила к постоянному выражению e предшествует проверка возможности его применения к этому выражению. Эта проверка осуществляется посредством вложенных друг в друга циклов (начиная с пустого слова как значения переменной для выражения) по длине значений переменных из выражения e', расположенного от начала правила до знака $\rightarrow$. Вложенность циклов определяется порядком первых вхождений в e' переменных для выражений. Длина значения каждой переменной не может превосходить длину постоянного перерабатываемого выражения.

В отличие от языка рефал-5, здесь вместо имен функций используются их определения. Знак равенства заменён стрелкой

вида $\rightarrow$ для рекурсивных правил и стрелкой с точкой вида $\rightarrow\cdot$ для заключительных правил (как это принято в записи нормального алгоритма Маркова).

Определение. *Условие применения правила имеет вид* <*функциональное выражение*>:<*выражение*>.

Здесь двоеточие читается как «типа». При этом <*выражение*> можно называть более длинно – образцом для сравнения.

Определение. *Алгоритмом Маркова-Турчина с охраняемыми правилами назовём простейший алгоритм Маркова-Турчина, в котором разрешено использование последовательности условий, начинающихся с запятой, применения любого правила, которые разделяются запятой. Эта последовательность должна заканчиваться непосредственно перед стрелкой. Каждая переменная из каждого функционального выражения, находящегося в одном из условий этой последовательности, должна встречаться до первой запятой в правиле или в одном из функциональных выражений из предшествующих условий в этом правиле.*

Примером последовательности с двумя условиями $e_1{:}e_3'B'e_4$ и $e_3e_4{:}e_2$ на применение правила с аргументом $e_1'A'e_2$ является $e_1'A'e_2, e_1{:}e_3'B'e_4, e_3e_4{:}e_2$.

Теперь организация работы вложенных друг в друга циклов по длине постоянных выражений, подставляемых вместо переменных в качестве их значения, распространяется и на выполнение всех условий, которые проверяются последовательно слева направо. При неудаче выполнения хотя бы одного условия длина записи значения соответствующей переменной цикла (переменные циклов упорядочены по первым вхождениям всех переменных для выражений, находящихся в образце для сравнения) увеличивается на наименьшее возможное положительное число. Если это невозможно, то увеличивается на такое же число длина записи значения первой (справа налево в заданном упорядочении) переменной, для которой это возможно (см. определение наращивания переменных при применении правила в языке рефал-5 [2, 6]). По существу такое использование последовательности условий является упрощённым и полиномиально эффективным вариантом отката (бэктрэкинга) языка пролог.

Определение. *Разрешим конец каждого правила (начиная со стрелки или с двоеточия) заменять на двоеточие с последующей записью алгоритма Маркова-Турчина с охраняемыми правилами. Разрешим также многократное и вложенное применение этого изменения. Полученный класс алгоритмов назовём классом алгоритмов Маркова-Турчина с древовидными правилами.*

В алгоритмах Маркова-Турчина с древовидными правилами (так же, как и в языке рефал-5) возможна запись вложенных условий, что может породить дерево условий. Последовательность условий с предшествующим выражением, выполнение которых записывается вычислением одного из заключительных функциональных выражений, назовём веткой условий для него.

Отметим, что если до $\rightarrow$ в записи ветки условий для правила в алгоритме Маркова-Турчина с охраняемыми правилами присутствует переменная, встречающаяся и ранее в этой ветке, то её значение совпадает со значением, полученным ранее. Поэтому в записи каждого из вспомогательных алгоритмов предпочтительно использовать новые имена переменных.

Введённое понятие алгоритма по существу, а не по форме является программой языка рефал-5.

Как и в языке рефал-5, константы, являющиеся непустыми словами, заключены в апострофы. Так, например, правило $e_1?e_2 \rightarrow [e_1e_2]$ (для стирания всех знаков ? вне текстов в структурных скобках) не будет осуществлено, так как символ ? не заключен в апострофы. Это правило должно быть записано в виде $e_1'?'e_2 \rightarrow [e_1e_2]$.

Пусть f – описание функции, полученное динамически (оно начинается и заканчивается соответственно левой и соответствующей ей правой фигурными скобками, находящимися в апострофах), x – исходные данные. Разрешим использование записи $\langle\langle f\rangle x\rangle$ как динамического интерпретатора.

В f все служебные символы взяты в дополнительные апострофы. Тогда в $\langle f \rangle$ сняты апострофы у каждого служебного символа, используемого в алгоритме Маркова-Турчина, и заключены в апострофы каждое слово без служебных символов. По существу происходит «раскодирование» выражения в запись программы. Обратную операцию, применённую к программе P, будем обозначать посредством $\#(P)$ и называть кодом её записи.

Дополнительно используем знак + вместо уже занятого в языке рефал знака ; (точка с запятой) для обозначения варианта композиции алгоритмов (обозначаемого обычно в традиционных процедурных языках программирования посредством знака ;).

Определение. *Алгоритмы, получившиеся в результате указанных расширений алгоритмов Маркова-Турчина с древовидными правилами, будем называть алгоритмами Маркова-Турчина.*

Использование операций $\langle\rangle$ и #() необходимо в связи с тем, что программы могут использоваться в двух кодировках: как субъект (когда она обрабатывает данные) и как объект (когда она является результатом работы другой программы). Операция $\#(P)$ превращает субъект P в объект (программу), а результат выполнения операций $\langle\#(P)\rangle$ является субъектом и совпадает с P. Результат выполнения операции $\#(\langle Q \rangle)$ совпадает с Q, если Q является кодом записи программы.

Обозначения

• Посредством $\| x \|$ обозначаем длину x.

• Знак $\cong$ означает условное равенство, используемое в теории алгоритмов и в теории сложности алгоритмов.

Примеры

1. Удвоение слова

$$\{e_1 \to\cdot\ e_1 e_1\}.$$

2. Результат применения встроенного интерпретатора к предыдущей программе и слову ab

$$\langle \{e_1 \to\cdot\ e_1 e_1\}'ab' \rangle = 'abab'.$$

3. Замена всех символов $\Rightarrow$ на $\to$ вне структурных скобок

$$\{e_1\ '\Rightarrow'\ e_2 \to [e_1\ '\to' e_2];\ \ e_1 \to\cdot\ \ e_1\}.$$

4. Универсальный алгоритм

$$\{(e_1)e_2 \to\cdot \langle\langle e_1\rangle\ e_2\rangle\}.$$

Видно, что использование встроенного интерпретатора позволяет записать программу универсального алгоритма весьма коротко.

Применение универсального алгоритма, обозначенного посредством U, к программе из примера 1 и слову ab имеет вид.

$$U((\{e_1 \to\cdot\ e_1 e_1\})'ab') =$$
$$= \langle \{e_1 \to\cdot\ e_1 e_1\}'ab' \rangle\ =\ 'abab'.$$

2. ОСНОВНЫЕ РЕЗУЛЬТАТЫ

Теорема 1. *Класс* **FP** *совпадает с классом всех рефал-5 функций, каждая из которых*

1) вычислима за полиномиально (от длины исходного аргумента) ограниченное сверху число выполненных её рекурсивных вызовов;

2) каждая ветка условий (включая заключительное функциональное выражение, расположенное после знака равенства) для каждого правила, определяющего эту функцию (включая функциональное выражение, на которое согласно этой ветке необходимо заменить исходный образец), содержит не более одного её рекурсивного вызова;

3) длина каждого постоянного (полученного в процессе вычисления) аргумента (находящегося между знаком $\to$ или $\to\cdot$ и разделителем – точкой с запятой или правой фигурной скобкой) рекурсивного вызова этой функции не превосходит длины исходного аргумента правила основной определяемой функции, сложенной с константой, одной и той же для всех применений этого правила;

4) все вспомогательные функции принадлежат классу **FP** *(в частности, метауниверсальная функция применяется только к аргументам, начинающимся с имён функций, принадлежащих* **FP***);*

5) является сборочной и бесстековой.

Доказательство. Число всех вызовов всех функций, использованных в определении исходной, удовлетворяющей сформулированным четырём условиям, не превосходит полинома от длины её исходного аргумента. В силу леммы 1, можно утверждать, что используемая память также не превосходит полинома от длины её исходного аргумента. Далее воспользуемся теоремой из [3] о том, что класс рефал-5 функций, вычислимых за полиномиальное число вызовов основной и вспомогательных функций и использующих полиномиально ограниченную память (измеряемую от длины исходного аргумента), совпадает с классом FP.

Работа машины Тьюринга, вычисляющей функцию из класса FP, может быть реализована рефал-5 функцией, удовлетворяющей сформулированным четырём условиям. Это связано с тем, что язык рефал-5 является обобщением нормальных алгоритмов, которые, в свою очередь, можно рассматривать как обобщение машин Тьюринга. При этом число шагов машины Тьюринга может быть лишь на единицу больше числа рекурсивных вызовов рефал-5-функции. Теорема доказана.

Формулировка доказанной теоремы для алгоритмов, обрабатывающих постоянные выражения языка рефал, не использует явно полиномиальное ограничение сверху на используемую память.

Определение. *Пусть M – целое положительное число. Алгоритм или данные назовём M-короткими, если длина их записи соответственно не превосходит M.*

Посредством U_M обозначим универсальный алгоритм, работающий над M-короткими программами и данными. То есть

$$\langle U_M((\#(P))x)\rangle \cong P(x)$$

для всех M-коротких программ P и M-коротких данных x.

Лемма 2. *Длина записи применимого на всех M-коротких программах и M-коротких исходных данных продолжения универсального для алгоритмов Маркова-Турчина алгоритма больше, чем $M - 46$.*

Доказательство от противного. Пусть имеется такое продолжение U_0 универсального алгоритма U_M, длина кода записи которого не превосходит $M - 46$. Тогда существует такая M-короткая программа P_0

$$\{e1 \to\cdot\ '1'\langle U_0(e1)e1\rangle\},$$

что для всех M-коротких данных x выполняется

$$\langle U_M((\#(P_0))x)\rangle \cong 1\langle U_0(x)x\rangle.$$

Правая часть этого условного равенства всегда применима к M-коротким данным x и, следовательно, его левая часть также всегда применима к M-коротким данным x. Полагая x равным $\#(P_0)$ получаем противоречие вида

$$\langle U_0(\#(P_0))\#(P_0)\rangle = 1\langle U_0(\#(P_0))\#(P_0)\rangle,$$
$$\text{если } ||\#(P_0)|| \le M.$$

Но $||\#(P_0)|| = ||\#(U_0)|| + 46$. Лемма доказана.

Теорема 2. *Длина записи алгоритма Маркова-Турчина, вычисляющего характеристическую функцию для проблемы применимости универсального (для M-коротких алгоритмов Маркова-Турчина и M-коротких исходных данных) алгоритма больше, чем $M - 270$.*

Доказательство от противного. Обозначим характеристическую функцию посредством g. Для неё имеет место

$$\forall x\, y\, (\|x\|, \|y\| \le M \Rightarrow$$
$$\Rightarrow (!\langle x\rangle(y) \Leftrightarrow g((x)y) = 1)).$$

Здесь и далее знак ! означает применимость.

Существование $(M-46)$-короткого алгоритма, реализующего вычисление выражения *if* $\langle(e1)e2\rangle = 1$ *then* $\langle U_M(e1)e2\rangle$ *else* 0, будет противоречить лемме 2, так как его можно записать в виде программы алгоритма Маркова-Турчина следующим образом

$$\{\{(e1)e2 \to\cdot \langle\langle\#(g)\rangle\ (e1)e2\rangle\ (e1)e2\} +$$
$$+ \{'1'(e1)e2 \to\cdot \langle U_M(e1)e2\rangle; e1 \to\cdot\ '0'\}\}.$$

Длина кода записи последнего алгоритма равна $170 + \|\#(g)\| + \|\#(U_M)\|$. Но U_M есть $\{(e1)e2 \to\cdot \langle\langle e1\rangle e2\rangle\}$. Длина его кода равна 54. Теорема доказана, так как

$$170 + 54 + 46 = 270.$$

Следствие 1 теоремы 2. *Проблема применимости универсального алгоритма Маркова-Турчина к данным алгоритмически неразрешима.*

Следствие 2 теоремы 2. *Невозможна алгоритмическая последовательность (зависящая от M) алгоритмов Маркова-Турчина, высисляющих характеристические функции для проблемы применимости универсального алгоритма Маркова-Турчина к любым M-коротким данным, обрабатываемых любыми M-короткими алгоритмами Маркова-Турчина.*

Доказательство методом от противного. В случае существования такой последовательности с помощью универсального алгоритма легко получается противоречие со следствием 1 теоремы 2.

Лемма 3. *Невозможен M-короткий алгоритм Маркова-Турчина, стирающий запись M-короткого алгоритма Маркова-Турчина, тогда и только тогда, когда последний не является записью стирающего свой код алгоритма, то есть*

$$\forall M \neg\exists B\,(||\#(B)|| \le M \;\&\; \forall A(||\#(A)|| \le M \Rightarrow$$
$$\Rightarrow (\neg(\langle A\#(A)\rangle = \Lambda) \Leftrightarrow \langle B\#(A)\rangle = \Lambda)\,)).$$

Доказательство от противного. Предположим, что для некоторого M найдётся M-короткий алгоритм B_0, такой что для всех M-коротких алгоритмов A верно

$$\neg(\langle A\#(A)\rangle = \Lambda) \Leftrightarrow \langle B_0\#(A)\rangle = \Lambda).$$

Взяв в качестве алгоритма A алгоритм B_0, получим противоречие

$$\neg(\langle B_0\#(B_0)\rangle = \Lambda) \Leftrightarrow \langle B_0\#(B_0)\rangle = \Lambda).$$

Теорема 3. *Проблема самостираемости M-коротких алгоритмов Маркова-Турчина не может иметь характеристическую функцию, задаваемую (M – 83)-коротким алгоритмом Маркова-Турчина.*

Доказательство проводится методом от противного. Существование алгоритма, вычисляющего выражение

$$if\ \ g(A) = 0\ \ then\ \Lambda\ \ else\ \ 1,$$

где g – характеристическая функция проблемы, сформулированной в теореме, противоречит лемме 3, если длина кода алгоритма не превосходит M. Но последний алгоритм может быть записан в виде алгоритма Маркова-Турчина

$$\{\{e1 \to\cdot\ \langle\langle\#(g)\rangle e1\rangle\} + \{'0' \to\cdot\ ;\ e1 \to\cdot\ '1'\}\}.$$

Длина его кода ровно на 83 больше, чем длина кода g. Теорема доказана.

Лемма 4. *Невозможен M-короткий алгоритм Маркова-Турчина, применимый к M-короткому алгоритму Маркова-Турчина, тогда и только тогда, когда последний не является применимым к своей записи, то есть*

$$\forall M \neg\exists B\,(||\#(B)|| \le M \;\&\; \forall A(||\#(A)|| \le M \Rightarrow$$
$$\Rightarrow(\neg!\langle A\#(A)\rangle \Leftrightarrow !\langle B\#(A)\rangle)\,)).$$

Доказательство проводится методом от противного. Аналогично доказательству леммы 3, вместо алгоритма A подставляем алгоритм B.

Теорема 4. *Существует константа, для которой проблема несамоприменимости M-коротких алгоритмов Маркова-Турчина не может иметь характеристическую функцию, задаваемую более коротким алгоритмом Маркова-Турчина, точнее*

$$\forall M \neg\exists C\,(\|\#(C)\| \le M - 103\ \&$$
$$\forall x(\|x\| \le M \;\&\; !C(x) \Rightarrow \forall A(\|\#(A)\| \le M \Rightarrow$$
$$\Rightarrow (\neg\,!\langle A\#(A)\rangle \Leftrightarrow \langle C\#(C)\rangle = 1))\,)).$$

Доказательство от противного. Существование алгоритма, вычисляющего выражение $if\ \langle C\#(A)\rangle = 1\ \ then\ \{e1 \to e1\}\ \ else\ \Lambda$, противоречит лемме 4, если длина его кода будет достаточно короткой.

Этот алгоритм можно реализовать посредством

$$\{\{e1 \to\cdot\ \langle\langle\#(C)\rangle e1\rangle\} +$$
$$+\ \{'0' \to\cdot\ \langle\{e1 \to e1\}\};\ e2 \to\cdot\ \}\}.$$

Длина его кода превосходит длину кода C ровно на 103. Теорема доказана.

Следствие теоремы 4. *Каково бы ни было положительное целое M, существует константа, для которой проблема самоприменимости M-коротких алгоритмов Маркова-Турчина алгоритмически неразрешима алгоритмом Маркова-Турчина, длина записи которого короче, чем M, на эту константу.*

Для замены несамоприменимости на самоприменимость достаточно к характеристической функции применить функцию, имеющую короткую запись и, следовательно, короткий код ее записи. А именно $\{'0' \to '1';\ '1' \to '0'\}$.

Литература

1. *Ахо А., Хопкрофт Дж., Ульман Дж.* Построение и анализ вычислительных алгоритмов. М.: Мир, 1979.

2. *Бабаев И.О., Герасимов М.А., Косовский Н.К., Соловьев И.П.* Интеллектуальное программирование. Турбо-Пролог и Рефал-5 на персональных компьютерах. Изд. СПбГУ, 1992.

3. *Косовский Н.К., Косовская Т.М.* Полиномиальный тезис Чёрча для рефал-5 функций, нормальных алгоритмов и их обобщений // Компьютерные инструменты в образовании, 2010. № 5. С. 12 – 21.

4. *Косовский Н.К.* Тезис Чёрча для полиномиальных по времени рекурсивных алгоритмов над словами и их длинами // Компьютерные инструменты в образовании, 2011. № 1. С. 4–11.

5. *Косовский Н.К.* Математические понятия алгоритма, основанные на языке рефал, для доказательств полиномиальной сложности вычислений // Материалы конференции «Информационные технологии в управлении» (ИТУ-2012). СПб.: ОАО «Концерн «ЦНИИ «Электроприбор», 2012. С. 84–92.

6. *Турчин В.Ф.* Рефал-5. Руководство по программированию и справочник / http: //www.refal.net/rf_5frm.htm (дата обращения 28.08.2012).

7. *Du D.Z., Ko K.I.* Theory of Computational Complexity. A Wiley-Interscience Publication. John Wiley & Sons, Inc. 2000.

8. Turbo Prolog reference guide. Borland Int. 1986.

Abstract

Refal5 functions and introduced by the author in [5] Markov-Turchin algorithms are under consideration in the paper. Such an algorithm is an essential modification of a Refal5 program. The test of its polynomial efficiency for the Turing machine implementation of such a program is offered. This test does not use the bound on the used memory size as it was stated in [3] for double polynomial Refal5 programs. The test is based on a special kind of recursion in a program which is an essential generalisation of a tail recursion proposed by T. Borland for a Turbo Prolog program. This generalisation is connected with the main Refal5 data type: a tree-like text structured by means of brackets. For a Markov-Turchin algorithm (with a built-in dynamical interpreter) some modifications of complexity theory theorems useful for mathematicians-programmers have simple proofs. In other words an introduced here mathematical notion of an algorithm modifies (mainly simplifying and combining into a single algorithm all needed auxiliary algorithms with the use of the built-in dynamical interpreter) the notion of a Refal5 function introduced by V.F. Turchin (see, for example, [2, 6]). This modification is, in particular, an extension of the Markov algorithm notion for tree-like formed texts with the help of structural brackets. The introduction of the built-in dynamical interpreter (in the terminology of V.F. Turchin – the extended built-in meta-universal function) computing an application of any dynamically received algorithm to the input data is an essential extension of the Markov algorithm. It is known that in practice algorithms and the initial data have bounded lengths. Theorems on variants of the halting problem for the Markov algorithm with such a restriction are proved in the paper. More precisely, lower bounds of the length of a Markov-Turchin algorithm (with a built-in dynamical interpreter) for the program checking the halting problem for short programs and short data are proved.

Keywords: Turing machine, polynomial number of steps, Refal5, Markov algorithm, complexity class **FP**, halting problem.

Косовский Николай Кириллович,
доктор физико-математических наук, профессор, заведующий кафедрой информатики математико-механического факультета СПбГУ,
kosov@NK1022.spb.edu.