Косовский Николай Кириллович
Косовская Татьяна Матвеевна

ПОЛИНОМИАЛЬНЫЙ ТЕЗИС ЧЁРЧА ДЛЯ РЕФАЛ-5-ФУНКЦИЙ, НОРМАЛЬНЫХ АЛГОРИТМОВ И ИХ ОБОБЩЕНИЙ

Введение

При разработке многократно используемых программ важно, чтобы их выполнение было достаточно быстрым. В [8] сформулирован обобщенный тезис Чёрча: „Функция, вычислимая за полиномиальное время на *разумной* вычислительной модели, использующей *разумное* измерение временной сложности, вычислима на детерминированной машине Тьюринга за полиномиальное время“ (то есть принадлежит классу **FP**[1,2]). Этот расширенный тезис Чёрча более естественно называть полиномиальным тезисом Чёрча для машин Тьюринга. „Разумность“ вычислительной модели разными исследователями понимается по-разному, следствием чего в последнее время всё чаще появляются „доказательства“ (содержащие ошибки) знаменитой проблемы: верно ли, что $\mathbf{P} = \mathbf{NP}$? При этом шаг различных моделей алгоритмов трактуется интуитивно.

Ниже доказано, что „разумность“ шага вычисления рекурсивной функции может заключаться в использовании только таких рефал-5-предложений из класса **FP**, длина записи результата выполнения которых (включая длину записи всего изменяемого стека) увеличивается не более чем на константу, единую для всех исходных данных (следствия 1 и 7). В этом смысле обычное определение общерекурсивной функции является „разумной“ вычислительной моделью, только если числа записываются в единичной системе счисления (только палочками). В этом случае принято [2] говорить о псевдополиномиальных вычислениях, причём псевдополиномиальные предикаты зачастую NP-полны.

Моделирование работы реальных алгоритмов на машине Тьюринга (что требуется для доказательства его полиномиальности по времени) кропотливо, утомительно и, зачастую, излишне. Оценки же числа шагов работы программ над конкретными данными, как правило, сделать не очень сложно, но это не всегда обеспечивает (обычно, в силу быстрого роста длины записи промежуточных вычислений) реальную полиномиальность по времени алгоритма. В [5] доказаны условия принадлежности классу **FP** паскалевидных функций, то есть функций, реализованных на модели, весьма близкой к языку программирования паскаль.

Достаточно сложным является подсчет числа шагов рекурсивного алгоритма. В настоящей статье доказаны необходимые и достаточные условия полиномиальности по времени функции, реализованной на языке программирования рефал-5 [7]. Поскольку этот язык является практически реализованным обобщением таких математических моделей алгоритма как нормальные алгоритмы, а также вводимые здесь алгоритмы Маркова-Поста и рекурсивные алгоритмы Маркова-Поста, то в качестве следствий основной теоремы доказаны условия полиномиальности по времени и этих математических моделей, применимых в теоретических исследованиях.

Язык рефал-5 (REFAL — Recursive Function Algorithmical Language) был разработан и реализован В.Ф. Турчиным в Нью-Йорке [7] (см., также [2]). Он предназначен для обработки исходных данных, записанных в поле зрения и в содержимом стека,

по существу являющегося стеком стеков. Понятие рефал-5-функции (т. е. функции, запрограммированной на языке рефал-5) может рассматриваться как математическое понятие алгоритма при условии, что не используются встроенные операторы работы над файлами.

Весьма актуальным является вопрос о возможности доказательства того, что программа, написанная на языке рефал-5, задает полиномиально быструю функцию, т. е. функцию, принадлежащую классу **FP** [1, 2, 4].

В статье вводится понятие дважды полиномиальных рекурсивных вызовов рефал-5-функции, обеспечивающее решение этого вопроса.

Следует отметить, что в [5] было введено понятие дважды полиномиальной паскалевидной функции (в том числе и над списком базовых подпрограмм, являющихся паскалевидными функциями), выполнение которого обеспечивает полиномиальную быстроту вычисления паскалевидной функции.

1. Основные определения и теорема

В последовательности условий языка рефал-5 в качестве разделителя, как и в языке пролог, используется запятая.

По сути дела, в языке рефал-5, в отличие от базисного рефала, используется дерево условий, так как выполнение каждого следующего условия зависит от результата проверки предыдущего условия, а условие может содержать сравнение с несколькими образцами, обеспечивающими ветвление.

Определение. *Рефал-5-предложением называется образец с переменными для частей аргумента, за которым следует знак равенства и функциональное выражение или следует дерево условий, начинающееся с запятой, на листьях которого после знака равенства находятся функциональные выражения, возможно использующие синтаксические (круглые) скобки, конкатенацию (приписывание друг к другу двух слов или функциональных выражений, или вызовов функций с аргументами в виде функциональных выражений).*

Рефал-5-функцией назовем следующее выражение: её имя, после которого находится заключённая в фигурные скобки последовательность рефал-5-предложений, отделённых друг от друга точкой с запятой.

Например,

ИМЯ_ФУНКЦИИ{ПРЕДЛОЖЕНИЕ_1;...;ПРЕДЛОЖЕНИЕ_N}

Под выполненными рекурсивными вызовами рефал-5-функции f для аргумента Q будем понимать все вызовы этой функции, выполненные в процессе вычисления функции f для аргумента Q.

Рефал-5-функция может содержать вызовы вспомогательных (определённых ранее и встроенных) функций.

Введем определения нескольких характеристик вычислительной сложности для рефал-5-функций. Пусть список S содержит все вспомогательные функции, достаточные для вычисления рефал-5-функции f.

Определение. *Под числом рекурсивных вызовов при вычислении рефал-5-функции f над списком функций S от аргумента Q понимаем число выполненных рекурсивных вызовов функции f при вычислении значения $f(Q)$, измеряемое относительно длины аргумента Q.*

Определение. *Под длиной записи результата выполнения рефал-5-предложения понимаем длину записи содержимого полученного стека, сложенного с длиной записи функционального выражения, находящегося на „листе“ выполненой „ветки“*

условий (в дереве условий) и в которое вместо всех переменных подставлены их значения, полученные в результате сопоставления с образцом из корня дерева и проверки всех условий, находящихся на выполненной „ветке“ условий.

Определение. *Под длиной промежуточных записей рефал-5-функции f над списком функций S понимаем функцию от длины записи исходных данных, равную максимальной длине записи аргументов и результатов у всех выполненных рефал-5-предложений и вызовов f, каждый раз сложенной с длиной записи всего содержимого стека в соответствующий момент.*

Вообще говоря, список S, являющийся параметром приведенных определений, может быть пустым. В этом случае слова „над списком функций S“ могут быть вычеркнуты как в этих определениях, так и в дальнейших формулировках.

Определение. *Под рефал-5-функцией с дважды полиномиальными рекурсивными вызовами над S понимаем такую рефал-5-функцию f, у которой как число рекурсивных вызовов при вычислении над S, так и длина промежуточных записей f над S не превосходят полинома от длины записи исходных данных, включающих в себя исходное содержимое всего стека.*

Следующая теорема усиливает и обобщает на список S теорему из [6], поскольку число шагов рефал-5-функции f над пустым списком не превосходит числа выполненных вызовов при вычислении f над пустым списком, умноженное на число выражений и всех их подвыражений в записи этой рефал-5-функции.

Теорема. *Класс всех рефал-5-функций с дважды полиномиальными рекурсивными вызовами над списком S, все функции которого принадлежат* **FP**, *совпадает с классом* **FP**.

Схема доказательства.

Во-первых, каждую функцию из **FP** можно реализовать дважды полиномиальной рефал-5-функцией, поскольку язык рефал-5 является обобщением нормальных алгоритмов, которые, в свою очередь, можно рассматривать как обобщение машин Тьюринга. При этом число шагов машины Тьюринга на единицу больше числа рекурсивных вызовов рефал-5-функции.

Во-вторых, рефал-5-функцию над списком S функций из **FP** можно считать схемой для реализации её машиной Тьюринга с оракулом, позволяющим вычислять каждую функцию из списка S. При этом длина записи промежуточных записей каждого вызова функции ограничена сверху одним и тем же полиномом от длины записи исходных данных.

Действительно, вычисление каждой из совместно рекурсивных рефал-5-функций может быть осуществлено на машине Тьюринга с оракулом-функцией традиционным образом с помощью отложенных вычислений, хранимых на дополнительной ленте, используемой в качестве стека. При полиномиальном от длины записи исходных данных числе вызовов функций длина этого стека будет ограничена сверху полиномом от длины записи исходных данных. Завершение рекурсии будет осуществлено за полиномиальное число шагов на машине Тьюринга с оракулом-функцией, позволяющем за один шаг вычислять любые (в конечном числе) вспомогательные функции. Всё это можно проделать дважды полиномиальным образом, измеряемым от длины записи исходных данных.

Как следует из [6], полученная функция будет принадлежать классу **FP**. Это завершает доказательство теоремы.

Следствие 1. *Класс всех рефал-5-функций с не более чем полиномиальным числом выполненных рекурсивных вызовов, при вычислении которых используются толь-*

*ко вызовы таких вспомогательных функций и такие рефал-5-предложения, которые принадлежат **FP** и могут увеличивать длину записи результата (и длину записи содержимого всего стека) не более, чем на константу (не зависящую от исходных данных), совпадает с классом* **FP**.

Действительно, единица, сложенная с числом осуществлённых вызовов f, не меньше числа выполненных рефал-5-предложений функции f. Каждое выполнение рефал-5-предложения (кроме последнего для функции) может увеличивать длину записи результата его применения не более, чем на константу, не зависящую от исходных данных. Следовательно, каждая функция, удовлетворяющая условию следствия, является функцией с дважды полиномиальными рекурсивными вызовами. Осталось применить теорему.

Это следствие обосновывает „разумность“ шага рефал-5-функции f как рекурсивного вызова f. Оно обеспечивает принадлежность классу **FP** функций, совершающих полиномиальное число таких шагов.

2. Нормальные алгоритмы

Определение. *Нормальный алгоритм, предложенный А.А.Марковым, — это последовательность подстановок, объединенная фигурной скобкой, вида*

$$\begin{cases} A_1 & \to [\cdot] \quad B_1 \\ & \vdots \\ A_k & \to [\cdot] \quad B_k \end{cases}.$$

Здесь запись $[\cdot]$ означает, что возможно как присутствие записи точки, так и её отсутствие. Подстановка, в которой присутствует точка, называется заключительной. Её выполнение означает, что после её применения алгоритм заканчивает работу.

В этом определении $A_1, ..., A_k, B_1, ..., B_k$ — слова (цепочки знаков, в том числе, возможно, и пустые цепочки) в некотором алфавите A.

Определение. *Интервалом вхождения слова P в слово Q будем называть рациональнозначный интервал с целочисленными концами, содержащий все номера букв слова Q, составляющие слово P.*

Например, имеется два интервала вхождения слова АБРА в слово АБРАКАДАБРА, а именно, интервалы (0, 5) и (7, 12).

Определение применения нормального алгоритма.

Слово P перерабатывается заданным нормальным алгоритмом следующим образом.

В слове P ищется первый (слева направо) интервал вхождения слова A_1 в P. Если такой интервал существует, то производится замена в этом интервале вхождения слова A_1 на слово B_1. Если такого интервала нет, то переходим к аналогичному поиску A_2.

...

Наконец, в слове P ищется первый (слева направо) интервал вхождения слова A_k в P. Если такой интервал существует, то производится замена в этом интервале вхождения слова A_k на слово B_k. Если и такого интервала не существует, то алгоритм заканчивает работу.

Если выполнилась подстановка, не являющаяся заключительной, то сначала запускаем процесс перерабатывания уже полученного результата подстановки. В противном случае алгоритм заканчивает работу.

Результатом работы алгоритма является результат последней выполненной подстановки или исходное слово P, если ни одна подстановка не выполнялась.

Выполнение каждой подстановки считается шагом нормального алгоритма. Таким образом, завершив переработку исходного слова алгоритм может совершить неотрицательное число шагов.

Определение. *Верхней границей временной сложности нормального алгоритма будем называть неотрицательную целочисленную функцию, значение которой для каждого n больше или равно максимальному числу шагов среди всех исходных слов длины n в алфавите A.*

Приведём пример нормального алгоритма, приписывающего слово УРА! перед любым словом.

$$\left\{ \quad \to \cdot \quad \text{УРА!} \right.$$

После нахождения интервала вхождения пустого слова в исходные данные (это будет интервал (0,1)) это вхождение будет заменено словом УРА! и алгоритм закончит работу.

Более сложно записывается следующий нормальный алгоритм приписывания слова УРА! в конец любого исходного слова.

$$\left\{ \begin{array}{rcl} qx & \to & xq \text{ для всех } x \in A \\ q & \to \cdot & \text{УРА!} \\ & \to & q \end{array} \right. .$$

Здесь используется дополнительная буква q, не входящая в алфавит A.

Первое правило последнего нормального алгоритма называют иногда метаправилом. На самом деле это список правил, количество правил в котором равно количеству букв алфавита A и эти правила расположены в записи нормального алгоритма в том же порядке, что и буквы алфавита A.

Пусть исходное слово УРА! и пусть {У,Р,А,!}$\subseteq A$. Тогда в соответствии с выписанным нормальным алгоритмом получаем следующую цепочку слов работы алгоритма по шагам.

$$\text{УРА!} \vdash \text{qУРА!} \vdash \text{УqРА!} \vdash \text{УРqА!} \vdash \text{УРАq!} \vdash \text{УРА!q} \vdash \text{УРА!УРА!}$$

Над исходным словом из 4-х букв алгоритм делает 6 шагов.

Следующие утверждения являются следствиями теоремы из раздела 1, но могут быть доказаны и как самостоятельные утверждения. Ниже приводятся два доказательства: прямое доказательство, не использующее результат теоремы, и доказательство, основанное на результате теоремы.

Следствие 2. *Класс функций, вычислимых нормальными алгоритмами за полиномиальное число шагов от длины записи аргумента, совпадает с классом* **FP**.

Прямое доказательство. Во-первых, нормальные алгоритмы являются обобщением машин Тьюринга. Действительно, команда машины Тьюринга (с внешним алфавитом $A = \{a_1, ..., a_n\}$) вида

$q_i a_j$	$\to$	$q_{i'} a_{j'}$	соответствует подстановке	$q_i a_j$	$\to$	$q_{i'} a_{j'}$,		
$q_i a_j$	$\to$	$q_{i'} R a_{j'}$	соответствует подстановке	$q_i a_j$	$\to$	$a_{j'} q_{i'}$,		
$q_i a_j$	$\to$	$q_{i'} L a_{j'}$	соответствует списку подстановок	$x q_i a_j$	$\to$	$q_{i'} x a_{j'}$	при	$x \in A$.

Осталось в конце списка подстановок добавить подстановки

$xq_i \to xq_ib$ при $x \in A$, $i \geq 1$,
$q_ix \to bq_ix$ при $x \in A$, $i \geq 1$,
$b \to$,
$q_0 \to \cdot$,
$xq_{-1} \to q_{-1}x$ при $x \in A$,
$q_{-1} \to bq_1$
$q_{-2}x \to xq_{-2}x$ при $x \in A$,
$q_{-2} \to q_{-1}b$
$\to q_{-2}$,

где q_1 и q_0 – соответственно начальное и заключительное состояния машины Тьюринга, a_j, $a_{j'}$ – буквы алфавита A (в котором работает нормальный алгоритм), q_i, $q_{i'}$ – состояния машины Тьюринга, b – бланковый символ из алфавита A, $i \geq 1$, $i' \geq 0$.

При этом число шагов нормального алгоритма увеличивается по сравнению с числом шагов машины Тьюринга не более, чем линейно.

Таким образом, класс **FP** входит в класс функций, вычислимых нормальными алгоритмами за полиномиальное число шагов.

Во-вторых, каждая подстановка нормального алгоритма может быть выполнена машиной Тьюринга за полиномиальное от длины записи перерабатываемого слова число шагов. Осталось объединить такого рода машины Тьюринга в одну, работа которой соответствует общей структуре работы нормального алгоритма.

На каждом шаге нормального алгоритма длина записи перерабатываемого слова увеличится не более, чем на $C = \max\{0, |B_1| - |A_1|, ..., |B_k| - |A_k|\}$. Следовательно, если длина записи аргумента равна n, то после выполнения i-го шага нормального алгоритма ($i = 1, ..., p(n)$, где $p(n)$ – количество шагов нормального алгоритма) длина записи перерабатываемого слова не превосходит $n + Ci$. Сумма полиномиального числа полиномов не превосходит некоторого полинома. Прямое доказательство завершено.

Доказательство следствия 2 теоремы из раздела 1. Используя переменные e_1 и e_2 для слов в алфавите A нормальный алгоритм вида

$$\begin{cases} A_1 & \to [\cdot] \quad B_1 \\ & \vdots \\ A_k & \to [\cdot] \quad B_k \end{cases}$$

можно записать в виде

$$\begin{cases} e_1A_1e_2 & \to [\cdot] \quad e_1B_1e_2 \\ & \vdots \\ e_1A_ke_2 & \to [\cdot] \quad e_1B_ke_2 \end{cases}.$$

При этом по переменной e_1 подразумевается цикл (длина e_1увеличивается на единицу начиная с нуля и до длины перерабатываемого слова).

Пусть первое правило содержит точку, а последнее нет. Тогда на языке рефал-5 последний алгоритм запишется в виде

$$NA\{e_1\ 'A_1\ 'e_2 = e_1\ 'B_1\ 'e_2; ...; e_1\ 'A_k\ 'e_2 = e_1\ 'B_k\ 'e_2 < NA\ \ e_1\ 'A_1\ 'e_2 >\},$$

где в угловых скобках записано, что алгоритм с именем NA применяется к $e_1\ 'B_k\ 'e_2$. Здесь если Q – пустое слово, то выражение $'Q'$ заменяется пустым словом. При каждом рекурсивном применении длина результата увеличивается не более, чем на константу. Следствие доказано.

3. Алгоритмы Маркова-Поста

Возможно обобщение нормальных алгоритмов до нормальных алгоритмов с детерминированным использованием однопосылочных правил Поста (сокращенно — алгоритмов Маркова-Поста). В этом случае подстановка заменяется однопосылочным правилом Поста вида

$$A_0 e_{j_1} A_1 e_{j_2} A_2 ... A_m e_{j_m} \to [\cdot] B_0 e_{i_1} B_1 e_{i_2} B_2 ... B_n e_{i_n},$$

где каждая из переменных для слов $e_{i_1}, e_{i_2}, ..., e_{i_n}$ встречается в левой части правила и $m \geq 0$, $n \geq 0$.

Определение. *Будем называть правило алгоритма Маркова-Поста правилом размножения переменной, если в его правой части для некоторой переменной имеется L различных вхождений, а в левой его части число различных её вхождений меньше L.*

Определение. *Детерминированное применение сформулированного правила Поста к слову Q заключается в организации самого внешнего цикла по длине e_{j_1} (начиная с нуля и кончая длиной слова Q), затем внутри этого цикла организации внутренних циклов последовательно по длине e_{j_2}, ..., длине e_{j_m} за исключением номеров, уже встретившихся ранее.*

Приведем пример алгоритма Маркова-Поста, осуществляющего удвоение слова.

$$\{e_1 \to \cdot\, e_1 e_1$$

Следствие 3. *Класс алгоритмов Маркова-Поста, не содержащих незаключительных правил размножения переменных и вычислимых за полиномиальное число шагов (выполненных правил) от длины записи аргумента, совпадает с классом* **FP**.

Прямое доказательство. Поскольку алгоритмы Маркова-Поста представляют собой обобщение нормальных алгоритмов, то очевидно, что совокупность всех их содержит класс **FP**.

Каждое детерминированное применение однопосылочного правила Поста без размножения переменных может быть промоделировано машиной Тьюринга за шагов, не превосходящее полинома от длины записи левой части правила и, следовательно от длины записи входного слова. Остаётся объединить все такие машины Тьюринга в одну, работа которой соответствует общей структуре работы нормального алгоритма с однопосылочными правилами Поста. Это возможно осуществить так, что она будет работать полиномиальное число шагов от длины записи исходного слова и любой промежуточный результат не будет превосходить по длине полинома от длины исходного слова. Прямое доказательство завершено.

Доказательство следствия 3 теоремы из раздела 1 базируется на тех же соображениях, что и её следствие 2.

Определение. *Алгоритм Маркова-Поста назовём дважды полиномиальным, если число его шагов и длина записи результата каждого шага его работы ограничены сверху полиномом от длины записи аргумента алгоритма.*

Следствие 4. *Класс дважды полиномиальных алгоритмов Маркова-Поста совпадает с классом* **FP**.

Это непосредственное следствие теоремы 1.

4. Рекурсивные алгоритмы Маркова-Поста

Дальнейшее обобщение алгоритмов Маркова-Поста связано с введением имён алгоритмов как определяемых, так и уже определённых ранее, называемых вспомогательными.

Определение. *Под словарным термом будем понимать терм на основе операции приписывания (конкатенации), которая не имеет обозначения ни префиксного, ни инфиксного, ни постфиксного и аргументы которой не берутся в скобки (они просто приписываются друг к другу). В качестве имен переменных для слов будем по-прежнему использовать букву e с нижними индексами.*

Определение. *Под функциональным словарным термом будем понимать терм на основе никак не обозначаемой операции приписывания и любых словарных функций в префиксной записи вида*

$$< \langle \text{Имя функции} \rangle \langle \text{Пробел} \rangle \langle \text{ Функциональный терм} \rangle > .$$

При этом имя функции не должно содержать пробел. Пробел может опускаться, если функциональный терм представляет собой пустое слово или начинается со знака <.

Пример использования такой записи имеется в конце раздела 2.

Определение. *Под правилом рекурсивного алгоритма Маркова-Поста будем понимать словарный терм, к которому приписан справа знак равенства „=“, к которому, в свою очередь, приписан справа функциональный словарный терм.*

Определение. *Правила будем разделять точкой с запятой „;“. Список всех правил будем заключать в фигурные скобки „{“ , „}“, перед которыми будем писать имя определяемого этим списком алгоритма. Список таких определений будем называть рекурсивным алгоритмом Маркова-Поста.*

Нетрудно заметить, что класс всех рекурсивных алгоритмов Маркова-Поста является подклассом всех алгоритмов базисного рефала.

Выполнение правила считается одним шагом.

Определение. *Пусть в правиле рекурсивного алгоритма Маркова-Поста после равенства количество различных вхождений каждой переменной для слов после знака = не более, чем количество её вхождений в этом правиле до знака=. Алгоритмы, все правила которых удовлетворяют этому условию, назовём алгоритмами без размножения переменных.*

Следствие 5. *Класс функций, вычислимых за полиномиальное число рекурсивных вызовов посредством рекурсивных алгоритмов Маркова-Поста без размножения переменных, в том числе и у вспомогательных функций, которые могут увеличивать длину записи результата на каждом шаге не более чем на константу, совпадает с классом* **FP**.

Прямое доказательство. Включение класса **FP** в класс рекурсивных алгоритмов Маркова-Поста, совершающих полиномиальное число вызовов, очевидно.

Так как для каждой вспомогательной функции в функциональных словарных термах алгоритма Маркова-Поста длина записи результата всегда ограничена сверху длиной записи аргумента, сложенной с константой, то длина записи любого промежуточного результата ограничена сверху полиномом от длины записи исходного аргумента, если число шагов вычисления каждой из определяемых функций ограничено сверху полиномом от длины записи исходных данных.

При этом если длина записи результата каждого шага алгоритма Маркова-Поста ограничена сверху полиномом от длины записи исходных данных для выполнения

этого шага, то полиномиальное число шагов такого алгоритма может быть промоделировано на машине Тьюринга с оракулом-функцией за полиномиальное число шагов (оракул нужен для вычисления вспомогательных функций). Прямое доказательство завершено.

Определение. *Рекурсивный алгоритм Маркова-Поста назовем дважды полиномиальным, если число его выполненных рекурсивных вызовов и длина записи результата каждого его шага не превосходят полинома от длины записи исходных данных.*

Следствие 6. *Класс дважды полиномиальных рекурсивных алгоритмов Маркова-Поста совпадает с классом* **FP**.

Теорема из [6] допускает следующую модификацию, обеспечивающую „разумность“ шага полиномиальных вычислений на машине Тьюринга с оракулом-функцией, удовлетворяющему ограничению из следствия.

Следствие 7. *Класс функций, вычислимых на машинах Тьюринга с оракулом-функцией, совершающих полиномиальное число шагов и оракулы которых принадлежат* **FP**, *а длины записи результатов работы этих оракулов не превосходят длины записи аргумента, сложенной с константой, совпадает с классом* **FP**.

Доказательство. Класс таких машин Тьюринга с оракулом является классом дважды полиномиальных машин Тьюрига с оракулом, поскольку длины всех промежуточных записей результатов их работы ограничены сверху полиномом от длины записи аргументов. Осталось применить теорему из [6]. Прямое доказательство завершено.

Это следствие 7 может также быть доказано как следствие из следствия 1.

Выполненный вызов оракула, удовлетворяющего условию из следствия 7, вполне может служить примером „разумного“ шага вычисления, полиномиальность числа которых обеспечивает принадлежность всего вычисления классу **FP**.

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. 167 с.

3. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982.

4. Косовская Т. М., Косовский Н.К. Основы доказательств полиномиальной быстроты простейших математических алгоритмов // Компьютерные инструменты в образовании. 2010 № 2. СПб. С. 3 - 13.

5. Косовская Т. М., Косовский Н.К. Принадлежность классу **FP** дважды полиномиальных паскалевидных функций над подпрограммами из **FP** // Компьютерные инструменты в образовании. 2010 № 3. СПб. С. 3 - 7.

6. Косовский Н. К. Условия полиномиальности алгоритмов, реализуемых на машинах Тьюринга с оракулами-функциями // XVII Международная школа-семинар

«Синтез и сложность управляющих систем» имени академика О.Б. Лупанова (Новосибирск, 27 октября – 1 ноября 2008 г.). Новосибирск: Изд-во Института математики, 2008. С. 70 – 74.

7. Турчин В.Ф. Рефал-5. Руководство по программированию и справочник. http://www.refal.net/rf_5frm.htm

8. Du D.Z., Ko K.I. Theory of Computational Complexity. – A Wiley-Interscience Publication. John Wiley & Sons, Inc. 2000. – 491p.