# УДК 004.432.42

# СИСТЕМА ТИПОВОГО КОНТРОЛЯ ПРОГРАММ НА ЯЗЫКЕ ФУНКЦИОНАЛЬНОГО ПРОГРАММИРОВАНИЯ FPTL[[1]](#footnote-1)

**© 2014 г. Бочаров И.А., Кутепов В.П., Шамаль П.Н.**

**Москва, МЭИ (национальный исследовательский университет)**

В статье описаны методы и алгоритм статического (до выполнения программы) контроля типов функциональных программ на созданном языке функционального параллельного программирования FPTL. Язык реализован на многоядерных компьютерах и успешно применяется на практике. FPTL является чисто функциональным языком, основанным на композиции функций путем применения четырех операций композиции функций и рекурсивных определений. Три из этих операций являются параллельными, что позволило создать достаточно эффективные алгоритмы параллельного выполнения программы на многоядерных компьютерах, используя интерпретацию программы. Это принципиально отличает FPTL от других функциональных языков программирования, основанных на лямбда-нотации (Haskell, ML и др.). Система типов данных в языке состоит из «встроенных» типов (real, int, bool и др.) и определяемых абстрактных типов данных. Функции в языке являются в общем случае полиморфными. В теоретической части статьи определены условия правильной типизации FPTL-программ, на основе которых разработан алгоритм контроля типовой правильности программы до ее выполнения. В существующей реализации языка на многоядерных компьютерах типовой контроль осуществляется в процессе выполнения программы, что занимает около 30% времени ее выполнения. Разработанная система типового контроля является статической и позволяет заметно сокращать время выполнения параллельных программ.

Ключевые слова: функциональные языки программирования, параллельные вычисления, типовой контроль.

**TYPE CHECKING SYSTEM FOR FUNCTIONAL PROGRAMMING LANGUAGE FPTL**

**© 2014 Bocharov I.A., Kutepov V.P., Shamal P.N.**

**Moscow, NRU MPEI**

The paper describes static type checking methods and algorithm for functional parallel programming language FPTL. FPTL is implemented on multicore systems and is widely used. FPTL is a pure functional language based on function compositions. By function composition we understand the application of four composition operations and recursive definitions. Three of composition operations are parallel. It made possible to implement rather effective algorithms of parallel program execution on multicore systems, using program interpretation. This is the key difference between FPTL and other functional languages, based on lambda-notation (such as Haskell, ML, etc.). Language type system consists of built-in types (int, real, bool, etc.) and user-defined abstract types. Functions in FPTL language are in general polymorphic. In theoretical section of this paper rules of correct typing and algorithm of type checking based on these rules are described. In the current language implementation on multicore system correctness of typing is checked while program is being executed. Type checking takes about 30% of time of program execution. Type checking system presented in the paper helps to significantly reduce time of program execution.

Keywords: functional programming language, parallel computing, type checking.

# Введение

Язык FPTL создавался с целью эффективного параллельного выполнения функциональных программ на компьютерных системах с общей памятью. В отличие от известных языков функционального программирования, таких как LISP[1], ML[2], Haskell[3,4], которые в большой степени основаны на лямбда-исчислении[5], FPTL основан на модели построения новых функций путем применения операций композиции функций и общей формы их задания в виде систем функциональных уравнений. Язык рекурсивных функций [6] – очевидный пример этого подхода к заданию вычислимых функций.

Кажущееся преимущество лямбда-основанных функциональных языков, обычно формулируемое как единообразный способ задания функций высших порядков, просто компенсируется в FPTL использованием в задании функций функциональных параметров. Этот традиционный способ «повышения» порядка функции допускает возможность в подставленной вместо функционального параметра функции снова использовать функциональные параметры, увеличивая таким образом функциональную иерархию в построении функций.

Однако, FPTL имеет ряд преимуществ перед основанными на лямбда-исчислении функциональными языками. Следующие особенности FPTL являются наиболее важными на практике:

* очевидная близость используемой нотации к традиционной математической форме задания функций,
* алгебраизированная форма задания функции, позволяющая извлекать ее композиционную структуру (схему функции), что существенно улучшает проектирование, анализ и эквивалентные преобразования функций [7,8,9],
* возможность явного отражения и варьирования параллелизма в функциональной программе, поскольку три из четырех операций композиции являются параллельными,
* простой алгоритм обнаружения параллелизма при выполнении, что позволяет исключить необходимость явного указания с помощью соответствующих процессных примитивов (как это имеет место, например в Haskell [4]), какие фрагменты программы могут выполняться параллельно,
* язык FPTL, как показывает практика, прост в освоении, позволяет широко использовать библиотеки, как правило, оптимизированных функций, вводить абстрактные типы данных, использовать целый ряд разработанных для него приложений, касающихся оценивания сложности параллельного выполнения программы, приведения ее к оптимальной параллельной форме, анализа структурной сложности программ [10] и др.

FPTL претерпел несколько реализаций. Одна из последних реализаций [11] оказалась недостаточно эффективной, так как она была выполнена на компьютерах с распределенной памятью (многоядерные компьютеры в то время еще не были широко распространены). Реализация FPTL на многоядерных компьютерах, напротив, оказалась успешной. Это подтверждают приведенные в [12] экспериментальные данные ускорения FPTL-программ.

В этой реализации, базирующейся на принципе интерпретации программы и динамическом выявлении и регулировании «зернистости» параллелизма и выполненной с использованием нитевого процессного параллелизма, можно достигать заметного ускорения при выполнении FPTL-программ [12].

FPTL – строго типизированный язык, однако контроль типовой правильности (корректности применения функций к их аргументам), который выполняется в данной реализации динамически, т.е. в процессе выполнения программы, увеличивает время ее выполнения на величину, составляющую до 30% от времени выполнения программы. Заметим, что продвинутые механизмы контроля типов присутствуют в компиляторах и интерпретаторах многих функциональных языков, включая компилятор языка Haskell GHC [13].

В настоящей статье описана разработанная система статического (до выполнения программы) определения её типовой корректности, что позволит существенно уменьшить время выполнения FPTL-программы.

В разделе 1 дано краткое описание языка FPTL и его особенностей. Разделы 2 и 3 посвящены заданию типов и условиям правильной типизации программ на языке FPTL, а в разделе 4 описан предложенный алгоритм типового контроля.

# 1. Язык FPTL

Язык FPTL (*functional parallel typified language*) создавался как прообраз общепринятой в математической практике композиционной формы задания функций в общем случае в виде систем рекурсивных функциональных уравнений, в которой при построении новых функций используется операция подстановки функций вместо функциональных переменных и условный оператор.

В FPTL введены четыре простые бинарные композиции функций, которые позволяют более тонко отражать композиционное представление функций и явно задают параллелизм при вычислении их значений.

Функции в FPTL рассматриваются как типизированные *(m, n)*-арные соответствия *(m ≥ 0, n ≥ 0)* между кортежами данных, где *m* - длина кортежа на входе функции, *n* - на выходе. Функции арности (0, 1) рассматриваются в FPTL как константы. Для *m* и *n*, равных 0, имеем кортеж нулевой длины, обозначаемый λ, со свойствами λα = αλ = α, где α - произвольный кортеж. Кортеж данных в языке представляется в виде последовательной записи его элементов.

В FPTL в отличие от общепринятой формы задания функций с явным указанием ее аргументов (так называемая форма задания общего значения функции) строго различается собственно функция, как отображение одного множества в другое, и ее аппликация к конкретным данным. Роль переменных в задании функций в FPTL выполняют функции выбора необходимого элемента из кортежа данных. Формально функция выбора аргумента, обозначаемая (в языке - просто ), при применении к кортежу произвольного типа данных длины выбирает его -й элемент, .

Для выбираемое значение есть . Функции в FPTL являются в общем случае частичными, причем неопределенное значение функции может быть выражено либо как неограниченный процесс вычисления ее значения, либо как специальное вычисляемое неопределенное значение, обозначаемое , со свойствами для любого кортежа .

Формально функции определяются как системы функциональных уравнений где – функциональные термы, построенные из заданных (базисных) функций и функциональных переменных путем применения четырех операций композиции функций: →, +, \*, •. Для функций и функциональных переменных задана их арность, а для базисных функций также их тип.

Пусть , заданные функции, а – определяемая новая функция, – обозначение кортежей. Синтаксис и семантика операций композиции определяется ниже. Сначала задается синтаксическая форма операции композиции, а затем её семантическое значение; символ обозначает равенство по определению.

1. Последовательная композиция (•):

; .

1. Операция конкатенации кортежей значений функций (\*):

; .

1. Операция условной композиции:

;

1. Операция объединения (графиков) ортогональных функций:

;

Напомним, что ортогональными называются функции и , такие, что для всякого кортежа данных α определена не более, чем одна из них. Например, общепринятая форма условного оператора в FPTL представляется как . Операция объединения ортогональных функций была введена с целью представления параллельных функций (известный пример – функция голосования в телефонии).

Все операции композиции обладают свойством ассоциативности, а операция объединения ортогональных функций является коммутативной. Приоритет операций композиции определяется следующим образом (в порядке возрастания): +, →, \*, •. Введение приоритета операций позволяет опускать часть скобок в записи функций.

При вычислении значений функций только операция последовательной композиции предполагает строгий порядок вычисления значений функций, к которым она применена. Остальные три операции композиции параллельны, причем для операций \*, → допустим произвольный порядок вычисления значений функций, к которым она применена. Для операции + только одновременное (или квазипараллельное) вычисление значений функций гарантирует получение результата.

Как уже было отмечено, термы в задании функций в виде системы функциональных уравнений представляют собой функциональные выражения, построенные из базисных функций и функциональных переменных путем применения операций композиции. Предполагается, что арности терма и определяемой им функциональной переменной в системе функциональных уравнений одинаковы. Функциональные переменные выполняют двойную роль при задании функции (построении функциональной программы): одни из них появляются как необходимые элементы при задании рекурсивных функций, другие определяются далее (в следующем уравнении уточняемой функции). Это позволяет просто реализовать пошаговую разработку функциональной программы по технологии проектирования «сверху-вниз», уточняя введенную функциональную переменную, определяя её в следующем функциональном уравнении.

Определяемые абстрактные типы данных в FPTL задаются по аналогии задания функций в виде систем реляционных уравнений. В FPTL можно использовать встроенные типы: *bool*, *real*, *int*, *string* и др. вместе с определенными в конкретной компьютерной системе операциями над ними.

Для определения новых типов данных используются конструкторы и деструкторы (обратные конструкторам функции). Для определения типов данных применяются те же операции композиции, которые применяются для задания функций, за исключением того факта, что операция + трактуется как операция объединения двух множеств данных. Используемые в определении абстрактного типа данных (системе реляционных уравнений) конструкторы и деструкторы выполняют роль базисных функций. В проведенных ранее исследованиях [9] доказано, что их достаточно для того, чтобы в FPTL можно было представить любую вычислимую функцию над абстрактными типами данных. При определении абстрактных типов данных можно использовать встроенные типы данных. Также в языке имеются конструкторы констант, позволяющие создавать данные встроенных типов.

Приведем пример пользовательского типа. Определим тип «стек комплексных чисел»:

**data** StackOfComplex{

Complex = real \* real.c\_complex;

StackOfComplex = c\_empty + (StackOfComplex \* Complex).c\_head;

}

В приведенном выше определении имеется три конструктора *c\_complex*, *c\_empty*, и *c\_head*, имеющие арности (2,1), (0,1) и (2,1) соответственно. В паре с определенными конструкторами определяются и деструкторы. Определения деструкторов извлекаются из определения типов и интерпретируются следующим образом:

,,.

В язык FPTL добавлена возможность задания параметризованных типов и функций. Приведем пример определения параметризованного стека, используя параметр ‘t (в FPTL обозначение типового параметра предваряется апострофом):

**data** Stack[‘t]{

Stack = c\_empty + (Stack[‘t] \* ‘t).c\_head;

}.

Приведем пример программы, вычисляющей количество элементов стека.

**data** StackOfComplex{

Complex = real \* real.c\_complex;

StackOfComplex = c\_empty + (StackOfComplex \* Complex).c\_head;

}

**scheme** Count {

Count = ~c\_empty → 0, ([1].~c\_head.[2].Count \* 1).add;

}

**application**

stack = (((c\_empty \* ((0.1\*0.1).c\_complex)).c\_head)\* ((0.1\*0.1).c\_complex)).c\_head;

%Count(stack);

FPTL также поддерживает импорт функций из внешних модулей. Для импорта функции из другой функциональной программы или библиотеки используется конструкция: **from** «имя модуля» **import** «имя функции»

Как отмечалось во введении, язык FPTL имеет несколько реализаций, и реализация FPTL на многоядерных компьютерах обладает рядом особенностей. Наиболее важной особенностью в контексте типового контроля является введение тернарной операции условной композиции взамен двух операций - и +, которые главным образом предназначены для описания условных конструкций. Новая тернарная операция – аналог условного оператора *if-then-else* в традиционных языках программирования и представляется в FPTL как (), где *p* – предикат, а f­1 и f2 – функции, значение одной из которых будет использовано в зависимости от того, истинно или ложно значение *p(x).*

Язык FPTL спроектирован так, что программирование на нем позволяет при помощи минимального набора средств, привычного для математической функциональной нотации, , не прибегая к введению дополнительных и обычно нарушающих общую архитектуру языка примитивов, отражать и «регулировать» параллелизм в программе.

# 2. Задание типов в FPTL

Напомним, что функции в FPTL задаются в виде систем в общем случае рекурсивных функциональных уравнений: где – определяемые функции (функциональные переменные), – термы, построенные применением описанных выше операций композиции к множеству базисных функций и функциональных переменных .

**Определение** Пусть – множество константных типов, которое состоит из множества встроенных в FPTL типов (*real*, *int*, *float*, *bool* и др.) и абстрактных (определяемых пользователем в FPTL) типов; – множество типовых переменных, таких, что каждая переменная принимает в качестве значений типы из . Пусть далее - множество кортежей длины , построенных из элементов множества . Очевидно, что состоит из единственного пустого кортежа, обозначаемого (как было указано выше) .

Кортеж типов длины *k* записывается следующим образом: .

Кортеж типов, не содержащий переменных, называется константным. Кортеж типов, содержащий переменные, интерпретируется как множество константных кортежей, полученных путем всевозможных подстановок константных типов из вместо вхождения в кортеж переменных типов.

**Определение** Содержательно тип (m, n)-арной функции определяется как множество однозначных отображений , где справа и слева от стоят константные кортежи. Множество однозначных отображений может быть как конечным, так и счетным.

Синтаксически тип функций задается в форме . В интерпретации эта запись представляет множество константных отображений, полученных всевозможными подстановками константных типов в типовые кортежи слева и справа от . Предполагается, что одинаковым переменным, входящим в кортежи, присваиваются одни и те же типовые константы при подстановке.

Требование однозначности в определении типа функции существенно, поскольку в случае, если оно не выполняется, мы получаем неопределенность. Приведем иллюстрирующий это пример: пусть тип равен Представим ситуацию, когда на входе имеется кортеж из двух целых чисел. Становится понятно, что мы не можем однозначно определить выходной типовой кортеж, поскольку неочевидно, ожидать нам на выходе целое или вещественное число. Однозначность не выполняется для функций, имеющих типы и .

**Определение.** Функция называется полиморфной, если входной или выходной кортеж хотя бы одного из ее типовых элементов содержит типовую переменную, или если тип функции содержит более одного типового элемента.

Функция выбора -го элемента из кортежа полиморфна и имеет тип .Базисная функция сложения полиморфна, поскольку ее тип состоит из более, чем одного типового элемента:

# 3. Условия правильной типизации функций в FPTL

Пусть функции представлены в форме систем функциональных уравнений вида. Будем исходить из предположения, что типы всех входящих в правые части базисных функций (встроенных функций, функций-конструкторов и деструкторов в определении абстрактных типов данных) заданы.

Введем отношение порядка на множестве типов, положив: . Отношение порядка на множестве кортежей вводится следующим образом: , если . Будем считать, что тип функции меньше или равен типу функции , если выполнено условие: для каждого типового элемента, принадлежащего существует типовой элемент , принадлежащий , такой, что: и . Положим, что .

Пусть - типовой элемент. Тогда типовой кортеж называется входным и обозначается . Аналогично, кортеж называется выходным и обозначается .

Для типа функции определим и как объединение множеств входных и выходных кортежей каждого типового элемента соответственно: .

Определим в введенных выше обозначениях условия корректной типизации для операций композиции.

1. – терм правильно типизирован, если ;

К условию правильной типизации для функций, полученных путем операции последовательной композиции, добавляется дополнительное условие, связанное с использованием функций выбора аргумента . Как было указано, эти функции полиморфны и выполняют тождественное преобразование между данными на i-м входе и выходе функции. Для функции , имеющей только константные типы, тип функции не является однозначным.

Поэтому особым образом определяется тип функций :

1. – правильно типизирован, если
2. – правильно типизирован, если .
3. – правильно типизирован, если ;

Определим условия строгой правильной типизации функций, заданных в виде системы функциональных уравнений. Эти условия вытекают из правил типизации функций, полученных применением операций композиции функций и необходимости равенства типов функциональных переменных и термов .

Добавим к условиям правильной типизации, определенных выше для операций композиции, еще два условия правильной типизации для функцииональных уравнений

1. Типы термов и определяемых ими функциональных переменных должны совпадать.
2. Все вхождения функциональной переменной в функциональные уравнения должны иметь одинаковый тип.

Заметим, что функция может быть неправильно типизирована (в определенном выше смысле), однако, при её применении к данным конкретного типа она, тем не менее, вычислит результат.

Перейдем к рассмотрению алгоритма типового контроля в FPTL.

# 4. Алгоритм типового контроля в FPTL.

По-прежнему исходим из общего задания функций в FPTL в форме системы функциональных уравнений . Считаем, что типы всех базисных функций заданы.

Перед проведением типового контроля производится ряд преобразований исходной системы функциональных уравнений:

Сначала для каждой функциональной переменной выполним для всех входящих в функциональных переменных подстановки вместо каждого вхождения в . Cтроим .

Повторяем процесс, строя путем подстановок вместо всех новых функциональных переменных, входящих в (переменных, которые появились в в результате осуществленных на первом шаге подстановок), соответствующих им термов.

Продолжаем этот процесс, пока для каждого не будут получены такие, что в не существует вхождений функциональных переменных, для которых ранее не выполнялась подстановка.

Полученную систему функциональных уравнений назовем приведенной. Полученная система функциональных уравнений эквивалентна исходной системе.

Затем представим каждый терм в эквивалентной форме: , где термы не содержат вхождений операции . Для этого используются следующие правила эквивалентности.

**Определение** Систему функциональных уравнений будем считать семантически корректной, если в представлении, полученном выше, в правой части для каждого терма существует по крайней мере один -терм , не содержащий вхождений функциональных переменных.

Требование семантической корректности на практике не является сильно ограничивающим, поскольку программы, которые не обладают этим свойством, могут выполняться бесконечно, не приводя к получению результата. Очевидно, что, в случае, если программа семантически корректна, процесс подстановки, описанный выше, не может длиться бесконечно.

В дальнейшем будем рассматривать только семантически корректные системы функциональных уравнений.

На третьем шаге работы алгоритма производится непосредственно типовая проверка.

После построения приведенной системы функциональных уравнений для семантически корректной программы в каждом уравнении имеется хотя бы один терм , такой, что он не содержит функциональных переменных. В случае, если в процессе построения приведенной системы уравнений получилось более одного слагаемого , не содержащего функциональных переменных, проверяем выполнение условий правильной типизации для операции +, т.е. равенство типов слагаемых.

Если это условие не выполняется, считается, что типизация не является правильной. Если же это условие выполняется в каждом уравнении приведенной системы уравнений, то полагаем тип равным типу одного из его слагаемых и для каждого вхождения функциональной переменной в соответствующее слагаемое определяем его тип. Если после этого все слагаемые в имеют тип, равный типу , считаем, что типизация рассматриваемой системы функциональных уравнений правильна. В противном случае считается, что правильность типизации нарушена.

В реализации языка FPTL на многоядерных компьютерах вместо двух операций композиции функций и + используется более эффективная с точки зрения параллельного выполнения программ тернарная операция, эквивалентная условному оператору в последовательных языках программирования. Эта операция записывается в форме , где – терм-условие. Легко показать, что она эквивалентна представлению . Поэтому перед применением алгоритма контроля типовой правильности функциональной программы все вхождения в нее термов, полученных путем применения тернарной операции, приводятся к данной эквивалентной форме.

Покажем работу алгоритма типового контроля на примере следующей программы:

Пусть

Производим подстановку вместо терм в правой части уравнения . Получаем. Все подстановки для сделаны, заметим, что в имеется терм, не содержащий функциональных переменных. Получаем, что .

В правой части уравнения нет терма, не содержащего вхождений функциональных переменных. Выполняем подстановки для: . В результате подстановки появился терм , не содержащий вхождений функциональных переменных. В соответствии с правилами определения типов для операции композиции получаем, что . Из этого следует, что тип функции равен .

Проверяем выполнение условий типовой корректности для всех слагаемых ви , содержащих вхождения функциональных переменных, считая их тип определенным ранее. . При проверке типовой корректности выражения мы встречаем ошибку типизации – тип функции равен как и тип функции равен . Делаем вывод о неверной типизации.

Пример можно изменить так, чтобы получить верную типизацию. Если все типы функций , то .

**Заключение**

Как уже ранее отмечалось, по предварительным экспериментальным данным разработанные метод и алгоритм статического типового контроля правильности программы позволяют уменьшать время выполнения программы на многоядерных компьютерах на величину до 30%.

ЛИТЕРАТУРА

1. McCarthy J. Recursive functions of symbolic expressions and their computation by machine. // Cambridge, Mass.: MIT, 1960.
2. Milner R.G. The standard ML core language. Polymorphism // The ML/LCF/Hope Newsletter 1985. V. 2 № 2.
3. Peyton Jones S. L. The implementation of functional programming languages. // London: Prentice Hall, 1987.
4. Tim Harris, Simon Marlow, Simon Peyton Jones. Haskell on shared-memory multiprocessor. Proceeding in Haskell workshop, 2005
5. Church A. The calculi of lambda-conversion // Ann. of Math. Studies, Princeton, N.J.: Princeton University Press, 1941. V. 6.
6. Клини С.К. Введение в метаматематику. М. ИЛ, 1957.
7. Кутепов В.П. Исчисление функциональных схем и параллельные алгоритмы. Программирование, 1976, №6.
8. Кутепов В.П., Фальк В.Н. Функциональные системы: теоретический и практический аспекты. Кибернетика, 1979, №1.
9. Кутепов В.П., Фальк В.Н. Направленные отноше­ния: теория и приложения. // Изв. РАН. Техн. ки­бернетика. 1994. № 4, 5.
10. Бажанов C.Е., Кутепов В.П., Шестаков Д.А. Структурный анализ и планирование процессов параллельного выполнения функциональных программ. // Изв. РАН. ТиСУ, 2005. № 6; Structural analysis and planning of processes of parallel execution of functional programs. // Journal of CSSI, 2005. V. 44 № 6.
11. Бажанов C.Е., Кутепов В.П., Шестаков Д.А. Язык функционального параллельного программирования и его реализация на кластерных системах. // Программирование. 2005, № 5.
12. Кутепов В.П., Шамаль П.Н. Реализация языка функционального параллельного программирования FPTL на многоядерных вычислительных системах. Доклад на конф. «Параллельные вычисления и задачи управления», Россия, Москва, 2012г.
13. <http://www.haskell.org/ghc>

REFERENCES

1. McCarthy J. Recursive functions of symbolic expressions and their computation by machine. // Cambridge, Mass.: MIT, 1960.
2. Milner R.G. The standard ML core language. Polymorphism // The ML/LCF/Hope Newsletter 1985. V. 2 № 2.
3. Peyton Jones S. L. The implementation of functional programming languages. // London: Prentice Hall, 1987.
4. Tim Harris, Simon Marlow, Simon Peyton Jones. Haskell on shared-memory multiprocessor. Proceeding in Haskell workshop, 2005
5. Church A. The calculi of lambda-conversion // Ann. of Math. Studies, Princeton, N.J.: Princeton University Press, 1941. V. 6.
6. Cleene C.K..Introduction to metamathematics. Moscow. IL, 1957.
7. Kutepov V.P. Ischislenie funkcional'nyh shem i parallel'nye algoritmy. Programmirovanie, 1976, №6.
8. Kutepov V.P., Fal'k V.N. Funkcional'nye sistemy: teoreticheskij i prakticheskij aspekty. Kibernetika, 1979, №1.
9. Kutepov V.P., Fal'k V.N. Napravlennye otnoshenija: teorija i prilozhenija. // Izv. RAN. Tehn. kibernetika. 1994. № 4, 5.
10. Bazhanov C.E., Kutepov V.P., Shestakov D.A. Strukturnyj analiz i planirovanie processov parallel'nogo vypolnenija funkcional'nyh programm. // Izv. RAN. TiSU, 2005. № 6; Structural analysis and planning of processes of parallel execution of functional programs. // Journal of CSSI, 2005. V. 44 № 6.
11. Bazhanov C.E., Kutepov V.P., Shestakov D.A. Yazyk funkcional'nogo parallel'nogo programmirovanija i ego realizacija na klasternyh sistemah. // Programmirovanie. 2005, № 5.
12. Kutepov V.P, Shamal P.N. Realizacija jazyka funkcional'nogo parallel'nogo programmirovanija FPTL na mnogojadernyh vychislitel'nyh sistemah. Doklad na konf. «Parallel'nye vychislenija i zadachi upravlenija», Moscow, Russia, 2012г.
13. <http://www.haskell.org/ghc>

1. Статья подготовлена в рамках выполнения проекта РФФИ №13-07-00810 [↑](#footnote-ref-1)