УДК 004.4

Особенности преобразования хвостовой рекурсии в функционально-потоковом языке параллельного программирования*

В.С. Васильев[a], А.И. Легалов[b], А.И. Постников[c]

Сибирский федеральный университет, пр. Свободный 82, Красноярск, Россия
[a]rrrFer@mail.ru, [b]legalov@mail.ru, [c]alpost@mail.ru
Статья получена 17.05.2013, принята 21.08.2013

В работе рассматриваются вопросы выявления хвостовой рекурсии и ее преобразования в цикл для языка функционально-потокового параллельного программирования Пифагор. Данный язык ориентирован на реализацию не только последовательных, но и параллельных рекурсивных вызовов. Вместе с тем, применение в нем хвостовой рекурсии также допустимо и позволяет реализовывать конвейерный параллелизм при использовании асинхронных списков. Среди возможных вариантов исследуется случай, когда рекурсивный вызов является последней операцией внутри задержанного списка, раскрываемой перед возвратом из функции. При этом предполагается что аргумент, поступающий на вход функции, является списком данных или скаляром. Данная ситуация является одной из простейших, но при этом широко применяется в различных программах. Реализация метода ее преобразования в итерацию позволяет в дальнейшем перейти к оптимизации программ, в которых хвостовая рекурсия сочетается с другими операциями, используемыми при функционально-потоковом параллельном программировании. Приведенный пример демонстрирует особенности замены хвостовой рекурсии специальной операцией повторения, позволяющей перенаправить выходной поток данных на вход, используемый для поступления аргумента функции. Для рассматриваемого случая предложен алгоритм преобразования хвостовой рекурсии в итерацию. Описаны действия, выполняемые оператором повторения функции при передаче данных на вход аргумента. Выполняемые алгоритмом преобразования осуществляются на реверсивном информационном графе, формируемом в процессе трансляции исходных текстов анализируемой функции.

Ключевые слова: хвостовая рекурсия, оптимизация программы, функциональное программирование, программирование потоков данных, информационный граф, разрушающее присваивание, задержанный список.

Features of tail recursion transformation in a functional dataflow language for parallel programming

V.S. Vasil'yev[a], A.I. Legalov[b], A.I. Postnikov[c]

Siberian Federal University, 82 Svobodny av., Krasnoyarsk, Russia
[a]rrrFer@mail.ru, [b]legalov@mail.ru, [c]alpost@mail.ru
Received 17.05.2013, accepted 21.08.2013

The paper deals with detection of a tail recursion and its transformation into the cycle for the PIFAGOR functional dataflow parallel programming language. The language is focused on the implementation of not only consistent calls but parallel recursive ones as well. At the same time, the use of the tail recursion is also acceptable and allows implementing pipelined parallelism while using asynchronous lists. Among the choices, the case where a recursive call is the last operation within the delayed list disclosed before returning from the function is studied. It is assumed that an argument entering the function is a data list or a scalar. This situation is one of the simplest and it is being widely used in various programs. The implementation of the method of its transformation into iteration allows further programs optimization where the tail recursion is combined with other operations used in a functional dataflow parallel programming. The examined example shows the features of replacing the tail recursion by a special repetition operation, which allows redirecting the output dataflow used for entering the function argument toward the input. For the case under consideration, the algorithm of the tail recursion transformation into iteration has been proposed. The actions made by the function loop statement in transferring the data to the argument input have been described. The performed transformations are carried out on a reversing information graph being formed in the process of translation of the analyzed function incoming texts.

Keywords: tail recursion, code optimization, functional programming, dataflow programming, information graph, destructive assignment, delayed list.

Введение. Одним из широко распространенных методов, используемых для оптимизации программ, является замена хвостовой рекурсии циклом. Данный подход применяется для различных языков программирования [1 – 3]. Особенно полезно его использование в языках функционального программирования, что позволяет избавиться от переполнения системного стека при большом числе итераций. Функционально-потоковый язык параллельного программирования

* *Работа поддержана грантом в рамках федеральной целевой программы «Научные и научно-педагогические кадры инновационной России» № 14.A18.21.0396*

Пифагор, как и другие функциональные языки, ориентирован на рекурсивное представление повторяющихся действий, поддерживая при этом не только последовательную, но и параллельную рекурсию [4]. Вместе с тем, написание программ с хвостовой рекурсией также допустимо и позволяет реализовать конвейерный параллелизм при использовании асинхронных списков [5]. Помимо этого, к циклам, порождаемым в ходе такой замены, можно применить дополнительные оптимизирующие преобразования, широко используемые в других языках программирования [6].

В языке Пифагор ветвление реализуется посредством задержанных списков, поэтому даже в функциях, обладающих хвостовой рекурсией, на путях между рекурсивным вызовом и оператором *return* могут встречаться операторы группировки в список и раскрытия задержанного списка. Кроме того, функционально-потоковый язык параллельного программирования Пифагор, в отличие от других функциональных языков, ориентирован на описание взаимосвязей между функциями с учетом управления по готовности данных. В связи с этими особенностями языка выделение хвостовой рекурсии, а также дальнейшая ее оптимизация, невозможны без анализа частных форм и поиска для каждой из них своего собственного алгоритма преобразования в итерацию. Обзор ряда ситуаций, сводимых к хвостовой рекурсии в рамках данного языка, приведен в [7], однако анализ и программная реализация представленных методов не проводились.

Целью статьи является формирование методов выявления хвостовой рекурсии и ее преобразования в циклические конструкции применительно к языку функционально-потокового параллельного программирования Пифагор.

Особенности преобразуемой функции. Транслятор языка Пифагор в качестве результата своей работы формирует промежуточное представление функционально-потоковой параллельной программы в виде реверсивного информационного графа (РИГ) [8], который используется для построения управляющего графа, задающего порядок выполнения вычислений, а также для анализа потоков данных функции и ее дальнейших оптимизационных преобразований. Поэтому задача выявления хвостовой рекурсии и ее преобразование к итерации сводится к задаче анализа и преобразования РИГ, который наряду с внутренним представлением в виде иерархии программных объектов имеет также внешнее текстовое представление. Помимо этого разработаны утилиты, позволяющие визуализировать данный граф с использованием графической библиотеки *graphviz* [9].

Особенности представления хвостовой рекурсии в языке Пифагор и ее последующего преобразования в итеративную версию можно рассмотреть на примере функции вычисления факториала. Основная функция *factTail* осуществляет накопление искомого произведения, промежуточное значение которого передается вместе с исходным числом в качестве аргумента, образуя пару *(N, P)*.

```
// factTail - факториал с хвостовой рекурсией      1
// X:1 – исходное значение для вычислений N        2
// X:2 – накопитель произведения P                 3
factTail<<funcdef X {                          //  4
   N << X:1;                                   //  5
   P << X:2;                                   //  6
   [((N,0):[=,>]):?]^(                         //  7
        P,                                     //  8
        { ((N,1):-,(P,N):*):factTail }         //  9
   ):. >> return                               // 10
}                                              // 11
```

Для ссылок на фрагменты данной функции используется нумерация строк, которая задана в однострочных комментариях, начинающихся с символов «//».

Окончательная функция *fact*, используемая для вычисления факториала, вызывает функцию *factTail* с хвостовой рекурсией, передавая в нее начальное значение аргумента *N*, определяющего значение вычисляемого факториала, а также начальное значение промежуточного произведения *P*, равное 1.

```
// fact – функция, вычисляющая значение факториала
// N – аргумент, задающий искомое значение факториала
fact << funcdef N {
     (N,1):factTail >> return;
}
```

На РИГ функции *factTail*, приведенном на рис. 1, видно, что оператор интерпретации, выполняющий рекурсивный вызов (узел 14), находится в задержанном списке (на рисунке задержанный список показан контуром, а в графе – узлом 4, задающим константу, ссылающуюся на узел 14). В программе данный задержанный список представлен в строке 9. Запуск операторов, расположенных в задержанном списке, произойдет только после операции его раскрытия. Следует также отметить, что функция *factTail*, определяющая рекурсивный вызов, является соответствующим аргументом оператора интерпретации, выполняющегося в этом списке последним.

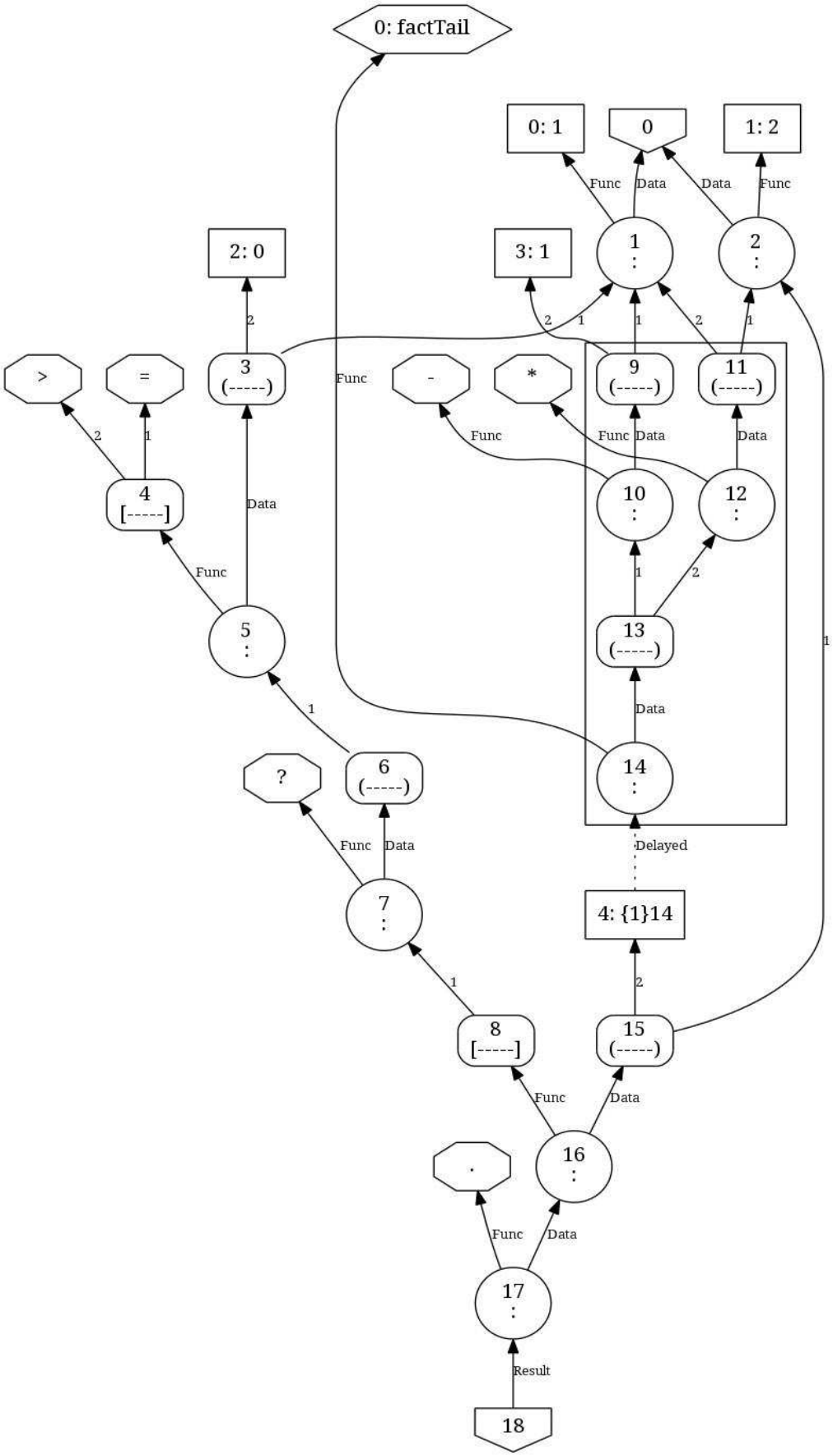

Рис. 1. РИГ функции factTail до оптимизации

Раскрытие задержанного списка [4] осуществляется оператором интерпретации (узел 17 РИГ, строка 10 функции *factTail*), функциональным аргументом которого является сигнальное (пустое) значение, обозначаемое точкой («.»). При этом предварительный выбор данного задержанного списка (без его раскрытия) из двух возможных альтернатив осуществляется при выполнении функции селекции в узле 16, формируемой в ходе анализа условий функцией «?» (узел 7). Узел 16 возвращает либо накопитель произведения, либо задержанный список с рекурсивным вызовом.

После раскрытия задержанного списка выполняется рекурсивный вызов, результат которого передается оператору возврата из функции «*return*».

Таким образом, результаты, полученные при рекурсивном вызове, помещаются в список данных. Затем над полученным списком выполняется функция, извлекающая элемент списка (в качестве функции выступает целое число, полученное оператором селекции), результат выполнения которой передается оператору *return*. Можно утверждать, что рекурсия приведенной функции является хвостовой [7] и может быть заменена итерацией.

Аналогично, любая функция, в которой между рекурсивном вызовом и возвратной вершиной расположены только операторы группировки в список и операторы интерпретации, на функциональный вход которых передается константа, будет содержать хвостовую рекурсию, а следовательно, может быть оптимизирована.

Преобразование хвостовой рекурсии в итерацию. Для оптимизации хвостовой рекурсии предлагается заменить рекурсивный вызов, используемый в качестве функционального аргумента операции интерпретации, на специальный управляющий аргумент *repeat*, задающий повторение вычислений в теле этой же функции.

При замене рекурсивного вызова *f*, выполняющего преобразование аргумента *X*, на функцию *repeat* осуществляется следующая модификация оператора интерпретации:

$$X{:}f \rightarrow X{:}repeat$$

При этом вновь вводимая функция повторения *repeat* должна выполнить следующие действия:

– при помощи повторного присваивания инициализировать аргумент уже выполняющейся функции *f* значением *X*;

– перевести все автоматы функции *f* в начальное состояние и уничтожить все значения ранее вычисленных величин (кроме значения *X*);

– восстановить начальные события функции *f*, обеспечивающие ее повторный запуск;

– повторно запустить обработчик событий функции *f*.

Преобразованный РИГ для функции *factTail* приведен на рис. 2. Вместо рекурсивного вызова появился функциональный аргумент *repeat*. Помимо этого возникла дополнительная информационная связь, определяющая зависимость между аргументом (узел 0) и оператором интерпретации (узел 14), обрабатывающим функцию повторения, что показано на рис. 2 пунктирной связью. Возможное обратное преобразование РИГ в исходный текст приведет к следующей функции вычисления факториала (строка 9).

```
// factTail - факториал с хвостовой рекурсией      1
// X:1 – исходное значение для вычислений N        2
// X:2 – накопитель произведения P                 3
factTail<<funcdef X {                          //  4
   N << X:1;                                   //  5
   P << X:2;                                   //  6
   [((N,0):[=,>]):?]^(                         //  7
        P,                                     //  8
        { ((N,1):-,(P,N):*):repeat }           //  9
   ):. >> return                               // 10
}                                              // 11
```

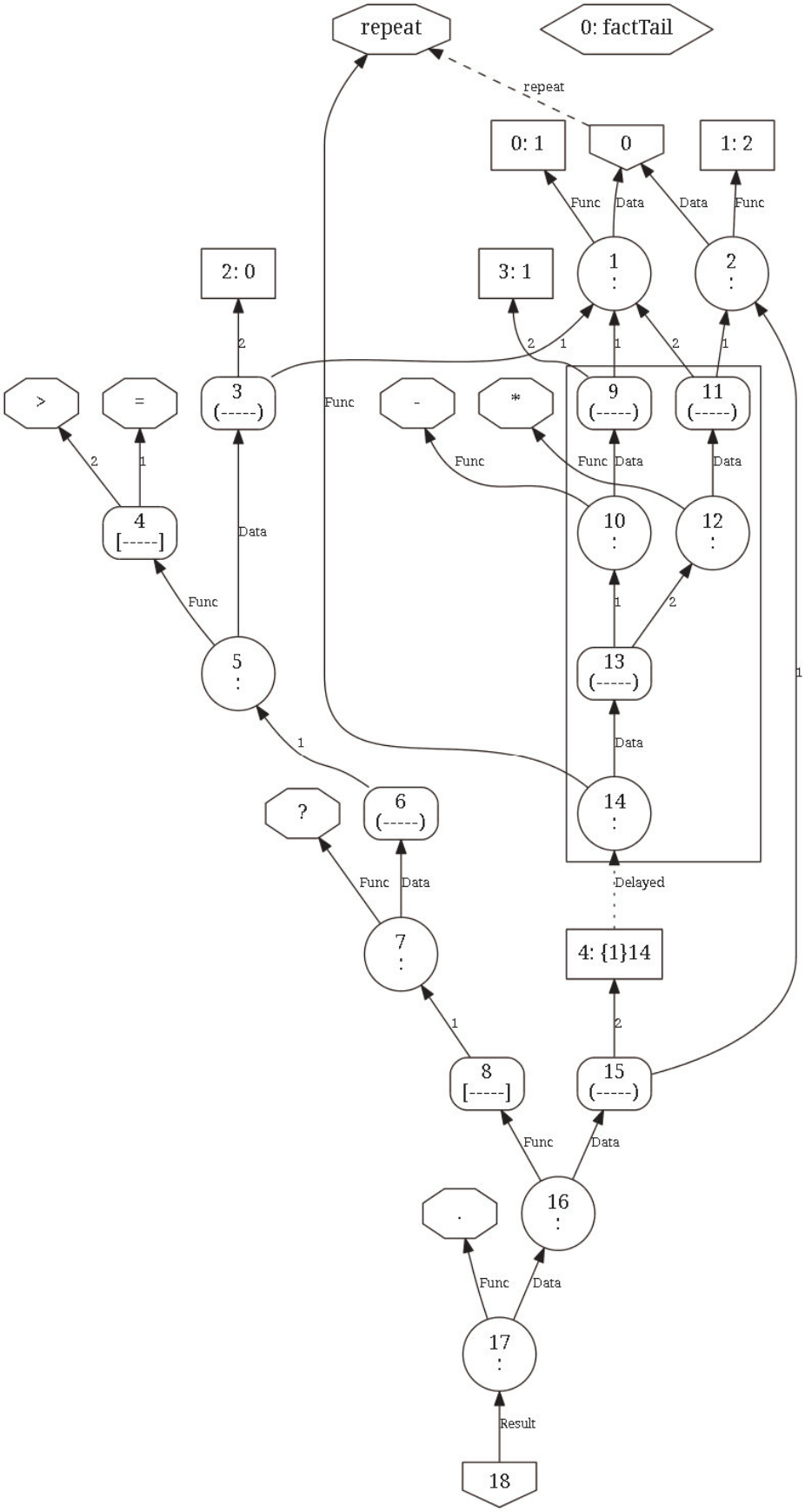

Рис. 2. РИГ оптимизированной функции factTail

Следует отметить, что непосредственное использование функции *repeat* при написании программ возможно, но не является надежным, так как непосредственное ее включение может привести к конфликтам и непредсказуемым ситуациям. Программист может неосознанно применить ее не только к допустимым хвостовым рекурсиям, но и в других ситуациях. Потребуется дополнительный анализ некорректных ситуаций при проведении верификации, которая в настоящее время разработана только для рекурсивных вычислений, позволяющих сконцентрировать внимание на логике программы и не рассматривать ресурсные конфликты, связанные с размещением данных в памяти, в связи с их невозможностью [10]. Поэтому в настоящее время использование функции *repeat* при написании программы не предусмотрено.

Заключение

В статье рассмотрен вариант реализации хвостовой рекурсии, который достаточно часто встречается при написании программ. Показано, каким образом можно определить, что рекурсия является хвостовой. Предложена функция повторения, позволяющая преобразовать рекурсию в цикл.

Описанный в работе метод оптимизации реализован и включен в модуль оптимизации программ языка программирования Пифагор [11, 12, 13].

Литература

1. Tail Recursion: [Электронный ресурс]. URL. http://c2.com/cgi-bin/wiki. TailRecursion (дата обращения: 01.04.13).
2. Tail Call Optimisation in Common Lisp Implementations [Электронный ресурс]. URL. http://0branch.com/notes/tco-cl.html (дата обращения: 01.04.13).
3. Tail recursion [Электронный ресурс]. URL http://www.haskell.org/haskellwiki/Tail_recursion (дата обращения: 01.04.13).
4. Легалов А.И. Функциональный язык для создания архитектурно-независимых параллельных программ // Вычислительные технологии. 2005. Т. 10, № 1. С. 71-89.
5. Легалов А.И., Редькин А.В., Матковский И.В. Функционально-потоковое параллельное программирование при асинхронно поступающих данных [Электронный ресурс] // Параллельные вычислительные технологии (ПаВТ'2009): тр. междунар. науч. конф. (Н. Новгород, 30 марта – 3 апр. 2009 г.). Челябинск: Изд-во ЮУрГУ, 2009. С. 573-578.
6. Ахо А.В., Лам М.С, Сети Р., Ульман Дж.Д. Компиляторы: принципы, технологии и инструментарий, 2-е изд.: пер. с англ. М.: И.Д. Вильямс, 2008. 1184 с.
7. Легалов А.И., Непомнящий О.В., Матковский И.В., Кропачева М.С. Преобразование хвостовых рекурсий в функционально-потоковых параллельных программах // Моделирование и анализ информационных систем. 2012. Т. 19, № 4. 2012. С. 48-58.
8. Легалов А.И. Использование асинхронных списков в потоковой модели вычислений / Третья сибирская школа-семинар по параллельным вычислениям / Томск: изд-во Томского ун-та. 2006. С. 113-120.
9. Описание графической библиотеки graphviz [Электронный ресурс]. URL: http://www.graphviz.org / (дата обращения: 01.04.13).
10. Кропачева М.С., Легалов А.И. Формальная верификация программ, написанных на функционально-потоковом языке параллельного программирования // Моделирование и анализ информационных систем. 2012. Т. 19, № 5. С. 81-99.

Reference

1. Tail Recursion [electronic resource]. http://c2.com/cgibin /wiki.TailRecursion (access date: 01.04.13).
2. Tail Call Optimization in Common Lisp Implementations [electronic resource]. Mode of access: http://0branch.com/notes/tco-cl.html (access date: 01.04.13).
3. Tail recursion [electronic resource]. Mode of access: http://www.haskell.org/haskellwiki/Tail_recursion (access date: 01.04.13).
4. Legalov A.I. A functional language for creating the architecture-independent parallel programs // Vychislitel'nye tekhnologii. 2005. T. 10. № 1. S. 71-89.
5. Legalov A.I., Red'kin A.V., Matkovsky I.V. The functional dataflow parallel programming for asynchronous incoming data // Parallel'nye vychislitel'nye tekhnologii Computing Technologies (PaVT' 2009): tr. mezhdunar. nauch. konf. Chelyabinsk: Izd-vo YuUrGU, 2009. S. 573-578. (Electronic edition).
6. Akho A.V., Lam M.S, Seti R., Ullman, J.D. Compilers: principles, technologies and tools. M.: "I. D.Williams", 2008. 1184 s.
7. Legalov A.I., Nepomnyashchy O.V., Matkovsky I.V., Kropacheva M. S. The transformation of tail recursion in functional dataflow parallel programs // Modelirovaniye i analiz informatsionnykh sistem. 2012. T 19. № 4. S. 48-58.
8. Legalov A.I. The application of asynchronous lists in the calculation dataflow model // Tret'ya sibirskaya shkola-seminar po parallel'nym vychisleniyam. Tomsk: zd-vo Tomskogo un-ta. 2006. S. 113-120.
9. The description of graphviz graphics library [electronic resource]. Mode of access: http://www.graphviz.org/ (access date: 01.04.13).
10. Kropacheva M.S., Legalov A.I. The formal verification of the programs written in a functional dataflow language of parallel programming // Modelirovaniye i analiz informatsionnykh sistem. T. 19. № 5. 2012. S. 81-99.