ДОКЛАДЫ АН ВШ РФ
2014 апрель–сентябрь № 2–3 (23–24)

ТЕХНИЧЕСКИЕ НАУКИ

УДК 004.052.42

ВЕРИФИКАЦИЯ ФУНКЦИОНАЛЬНО-ПОТОКОВЫХ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ МЕТОДОМ ИНДУКТИВНЫХ УТВЕРЖДЕНИЙ

Ю.В. Удалова, А.И. Легалов
Сибирский федеральный университет

Метод индуктивных утверждений изначально был сформулирован для верификации последовательных императивных программ. В статье предложено его использование для верификации функционально-потоковых параллельных программ на языке Пифагор. Представленная адаптация метода опирается на информационный граф функционально-потоковой модели параллельных вычислений. Спецификация пользователя задается с применением классических начальных и промежуточных утверждений. Для описания спецификации применяются формулы, которые реализованы с применением специализированного языка. Начальное утверждение описывает общий вид аргумента функции. Для его описания доступны константы, определяющие тип аргумента, а также формулы, определяющие принадлежность аргумента к указанным интервалам, если он является численным. Промежуточные утверждения ставятся в соответствие с узлами информационного графа программы. Для описания промежуточных утверждений доступны все базовые операторы языка Пифагор, а также формулы, применяемые к начальному утверждению. Это позволяет накладывать на выполняемую программу различные условия и ограничения. Условия рекомендуется формулировать таким образом, чтобы они возвращали булево значение. В этом случае можно однозначно сделать заключение о том, соответствует или нет выполнение программы заданному утверждению. Помимо этого, полученное булево значение используется для отображения ошибок, возникающих в программе. Эти ошибки отображаются на информационном графе программы. Применение метода индуктивных утверждений для верификации функционально-потоковых параллельных программ позволяет проверить соответствие вычислений программы спецификации пользователя и интервально оценить вычисленные программой результаты. Это облегчает процесс разработки, тестирования и отладки функционально-потоковых параллельных программ.

Ключевые слова: верификация, спецификация, корректность, функционально-потоковое параллельное программирование.

Введение

Современное параллельное программирование в основном опирается на императивную парадигму. Вместе с тем развиваются и другие альтернативные подходы, в частности функционально-потоковое параллельное (ФПП) программирование [1], ориентированное на разработку архитектурно-независимых параллельных программ. Применение функциональной парадигмы, с одной стороны, облегчает создание параллельных программ, акцентируя внимание в большей степени на потоках данных и в меньшей – на потоках управления. С другой стороны, необходимо использовать новые методы отладки и верификации.

Основные методы отладки ФПП программ базируются на различных стратегиях обхода информационного графа [2]. Для повышения эффективности этого процесса разработаны соответствующие инструментальные средства, обеспечивающие использование специальных формул для спецификации проверяемых условий [3]. Дальнейшее развитие работ, представленное в статье, связано с инструментальной поддержкой методов верификации ФПП программ. Для этого используется метод индуктивных утверждений, изначально ориентированный на

императивное последовательное программирование [4]. Адаптация метода к ФПП языку связана с его применением на информационном графе программы. Введены специальные формулы для спецификации, предоставлена возможность расстановки утверждений спецификации на информационном графе ФПП программы. Метод реализован в инструментальной среде, предназначенной для разработки, отладки и верификации ФПП программ.

1. Особенности верификации ФПП программ

Особенности модели ФПП вычислений, такие как независимость вычисленных значений от порядка обхода операторов, наличие неявных механизмов синхронизации и отсутствие ресурсных конфликтов, позволяют использовать для верификации методы доказательства теорем [4], к которым относится и метод индуктивных утверждений. Применение метода индуктивных утверждений базируется на знании порядка обхода операторов программы. Для ФПП программы обход вершин-операторов производится в соответствии с зависимостью между информационными связями. Верификация на графе обладает таким достоинством, как сужение множества исходных утверждений, используемых для построения или проверки последующих утверждений. Это связано с тем, что утверждения, получаемые при рассмотрении конкретной вершины-оператора, опираются на предыдущие не от каждого ранее рассмотренного оператора программы, а от множества вершин-операторов, связанных с рассматриваемой.

Для спецификации ФПП программ выбран формат, в котором обязательно наличие входного утверждения, а промежуточные и целевое утверждения не обязательны. Промежуточные утверждения предлагается приписывать к операторам-вершинам графа. Выбор такого формата спецификации обусловлен тем, что модуль верификации основан на методе индуктивных утверждений, а также возможностями визуализации ошибочных вычислений на графе, повышающими эффективность локализации некорректных вычислений.

Спецификация исходных данных (аргументов функций) основана на применении конкретных значений и дополнительных формул. Введены специальные обозначения для различных типов данных и операций:

~unknownnumber – неизвестное число,

~unknownbool – неизвестное логическое значение (ложь или истина),

~unknown – неизвестное значение, которое может быть числом, текстом, логическим значением, списком, строкой или любым другим элементом данных, а также являться результатом верификации программного оператора над данными, не определенными во множестве правил для верификации программы,

~gt A – число большее, чем указанное число A,

~lt A – число меньшее, чем указанное число A,

~ge A – число большее либо равное указанному числу A,

~le A – число меньшее либо равное указанному числу A,

~A interval B – число, лежащее в указанном интервале [A, B].

Например, начальное утверждение может выглядеть как

```
(true, ~unknownbool, ~gt 0),
```

где аргумент является списком данных, первый элемент которого равен истине, второй является булевой величиной, а третий – числом большим нуля.

Промежуточные утверждения прикрепляются пользователем к произвольным вершинам-операторам графа и являются выражениями на языке Пифагор, способными включать также вышеописанные конструкции и дополнительные обозначения:

ARG – аргумент функции,

NODE – значение той вершины-оператора графа функции, к которой добавлено пользовательское условие,

NODE<натуральное число>– значение оператора с указанным номером. Номера назначаются операторам автоматически перед началом верификации и видны пользователю.

Таким образом, промежуточное утверждение может быть, например таким:

((NODE,ARG):<, (NODE,~0 interval 1):!=):*,

т. е. значение текущего оператора меньше, чем аргумент функции и не совпадает с интервалом (0,1). Пользователь может писать произвольные формулы в спецификации, но рекомендуется выстраивать их так, чтобы они возвращали булевы значения.

Процесс верификации опирается на базу правил, определяющих срабатывание операторов языка Пифагор [1], и заключается в обходе графа программы, в ходе которого вычисляются значения операторов и промежуточных утверждений спецификации. При этом используются формулы, представленные в описанной выше нотации. Если промежуточное утверждение возвращает истину, то это выделяется цветом на графе и считается, что ожидание пользователя о свойстве выполнения программы подтвердилось. Возврат ложного значения интерпретируется как опровержение спецификации пользователя и отмечается на графе другим цветом. Результатом верификации оператора или формулы спецификации может оказаться значение ~unknown, если в базе правил соответствие не найдено. Это показывает, что автоматическая верификация не смогла ни подтвердить, ни опровергнуть соответствие программы заданной спецификации. Смысл других значений, полученных при верификации формул спецификации, определяется пользователем самостоятельно.

Возможно проведение верификации только при задании начального утверждения (общего вида аргумента функции), т. е. при отсутствии промежуточных утверждений. В этом случае польза верификации состоит в получении интервальных оценок результатов срабатывания операторов.

2. Примеры верификации ФПП программ

ФункцияAbs получает число P и вычисляет его модуль.

```
Abs <<funcdef P{
({P:-}, P): [(P,0):(<,>=) :?]:. >>return
}
```

Информационный граф функции приведен на рис. 1, *а*.

Пусть спецификацией начальных данных функции является ~lt 0, т. е. аргумент P, это число меньшее нуля. Результат верификации функции для указанной спецификации представлен на рис. 1, *б*.

Оператор с номером 1 {P:-} (рис. 1, *а*) при подстановке P станет {~lt0:-}. Поиск соответствия в наборе правил даст результат {~gt0}, т. е. задержанный список с положительным числом.

Оператор под номером 2 ({P:-}, P) при подстановке P и вычисленного для предыдущего оператора значения даст ({~gt0}, ~lt0).

Оператор 3 (P,0) станет (~lt0, 0). Команда (P,0):(<,>=) заменится на (~lt0, 0) :(<,>=), это означает проведение двух сравнений ~lt0<0 и ~lt0>=0, поиск в наборе правил выдаст результат true и false, что сформирует список (true, false).

Оператор 5 (P,0):(<,>=):? после подстановки предыдущих вычисленных значений станет равен (true, false):?. Оператор 6 (true, false):? не содержит данных,

содержащих утверждения спецификации, поэтому его значение не ищется в наборе правил, а вычисляется интерпретатором. Оно равно [1], это параллельный список из индексов истинных значений в списке.

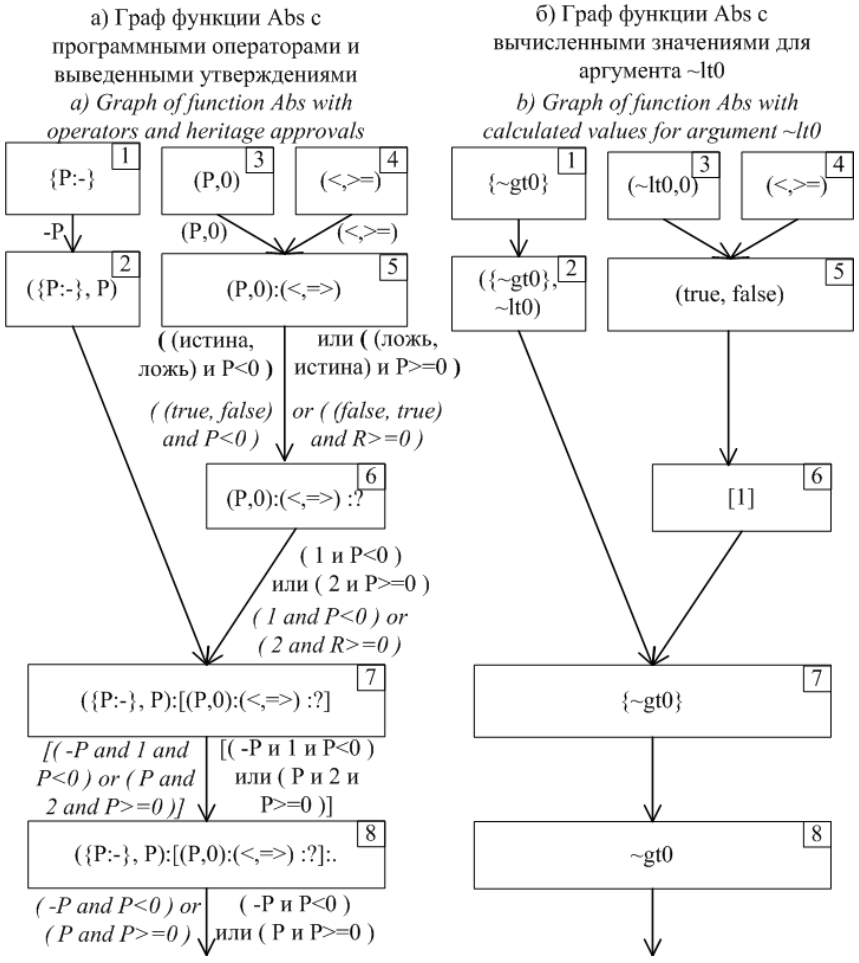

Рис. 1 – Информационный граф функции Abs (а) и вычисленные значения на графе функции Abs для аргумента P=~lt0 (б)

Fig. 1 – Information graph of function Abs (a) and calculated values on the graph of function Abs for argument P=~lt0 (b)

Оператор 7 ({P:-}, P): [(P,0):(<,>=) :?] после подстановки вычисленных значений станет ({~gt0}, ~lt0) : [1], это выбор первого элемента из списка, что даст {~gt0}.

Оператор 8 ({P:-}, P): [(P,0):(<,>=) :?]:. при подстановке значений, вычисленных предыдущими операторами, станет равен {~gt0}:., что даст результат ~gt0.

Таким образом, верификация над указанным начальным утверждением ~lt0 показала, что функция Abs, получающая на входе отрицательное число, вычисляет число положительное.

Функция Abs не содержит вызовов других пользовательских функций или рекурсий, все ее операторы вычисляются за один шаг верификации. Кроме того при ее верификации не использовались пользовательские условия на графе.

Рассмотрим рекурсивную функцию Add, которая получает список произвольной длины v и вычисляет сумму его элементов.

```
Add<<funcdef v {
 //сравнение длины списка с единицей
[((v:|,1):[<,=,>]):?]^ (
{.},      //пустой список
      {v:1},    //первый элемент из списка
    /*сумма первых двух элементов списка и список
        v после удаления двух его первых элементов
        подаются на вход рекурсивной функции Add*/
{((v:1,v:2):+, v:-1:-1:[]):Add:.}
  ):.>>return;}
```

Упрощенный информационный граф функции, отражающий основные операторы программ, приведен на рис. 2.

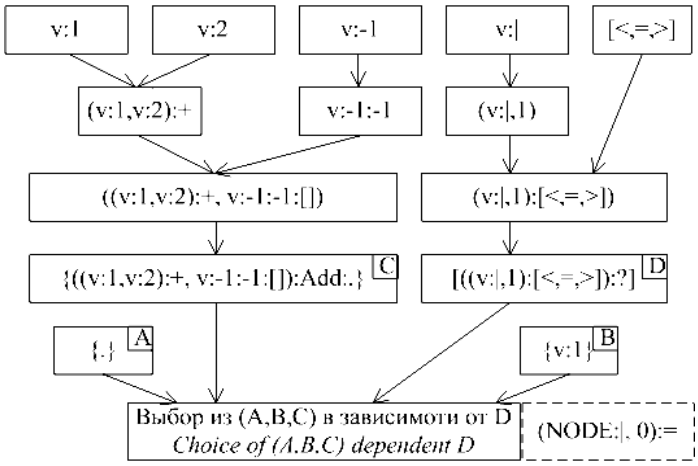

Рис. 2 – Сокращенный информационный граф функции Add

Fig. 2 – Reduced information graph of function Add

К заключительному оператору графа приписано пользовательское условие (NODE:|, 0):=, отражающее требование, что результатом работы функции является не список, а значение. Требование истинности данного условия ожидается пользователем на завершающей итерации рекурсии. На рис. 2 условие пользователя отмечено пунктирной границей.

Пусть начальные данные определены как (~ –1 interval 1, ~3 intreval 4, ~ge1). То есть подается список из трех аргументов, первый из которых лежит в интервале [–1, 1], второй находится в интервале [3, 4], а третий больше или равен 1. В процессе верификации будут рассмотрены все (для указанного аргумента три) итерации рекурсии Add, пользовательские условия вычисляются каждый раз заново на каждой итерации рекурсии.

Ход верификации функции Add приведен на рис. 3, 4, 5. Полученные на последней итерации значения операторов error не относятся к верификации, а являются обрабатываемыми константами языка Пифагор.

Проведенная верификация покажет, что условие пользователя выполняется по завершению рекурсивной функции и вычисленное функцией значение для входных данных (~ –1 interval 1, ~3 intreval 4, ~ge1) будет ~ge3, т. е. число больше

либо равное трем. Таким образом, верификация для заданного аргумента подтвердила требование пользователя о виде результата функции и дала ему интервальную оценку.

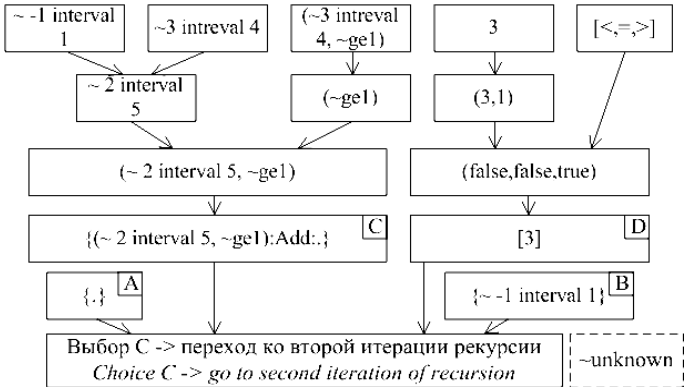

Рис. 3 – Первая итерация рекурсивной функции Add
Fig. 3 – First iteration of recursive function Add

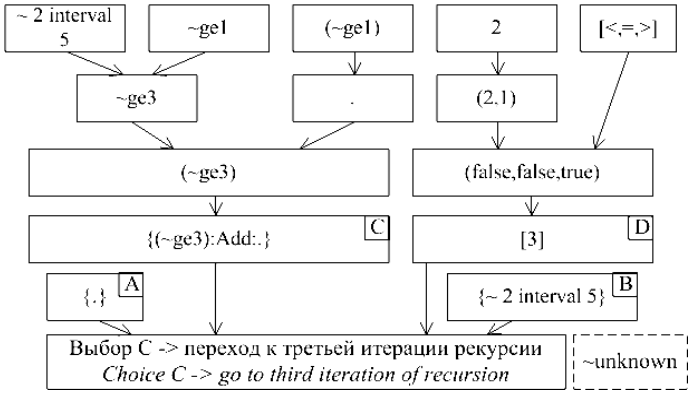

Рис. 4 – Вторая итерация рекурсивной функции Add
Fig. 4 – Second iteration of recursive function Add

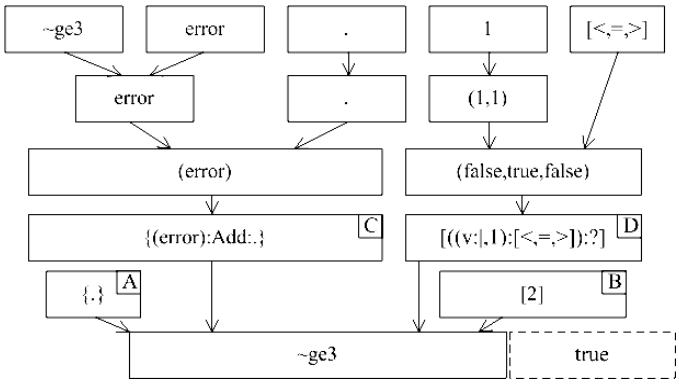

Рис. 5 – Третья итерация рекурсивной функции Add
Fig. 5 – Third iteration of recursive function Add

Заключение

Представлен метод формальной верификации ФПП программ на языке Пифагор, основанный на применении метода индуктивных утверждений к информационному графу ФПП программы, предполагающий спецификацию входных данных программы с помощью конкретных и интервальных значений, оперирующий пользовательскими условиями, накладывающими требования на промежуточные результаты вычислений программы. Метод обеспечивает автоматизированную верификацию ФПП программы на языке Пифагор для обобщенного множества входных данных и позволяет установить соответствие вычислений, выполняемых программой, спецификациям разработчика, а также получить интервальные оценки вычисляемых значений.

ЛИТЕРАТУРА

1. **Легалов А.И.** Функциональный язык для создания архитектурно-независимых параллельных программ // Вычислительные технологии. – 2005. – Т. 10, № 1. – С. 71–89.
2. **Удалова Ю.В., Легалов А.И., Сиротинина Н.Ю.** Средства отладки функционально-потоковых параллельных программ // Доклады Академии наук высшей школы Российской Федерации. – 2008. – № 1 (10). – С. 96–105.
3. **Удалова Ю.В., Легалов А.И., Сиротинина Н.Ю.** Методы отладки и верификации функционально-потоковых параллельных программ // Журнал Сибирского федерального университета. Серия: Техника и технологии. – 2011. – Т. 4, № 2. – С. 213–224.
4. **Лисков Б., Гатэг Дж.** Использование абстракций и спецификаций при разработке программ. – М.: Мир, 1989. – 424 с.

VERIFICATION OF FUNCTIONAL DATAFLOW PARALLEL PROGRAMS USING THE METHOD OF INDUCTIVE ASSERTIONS

Udalova J.V., Legalov A.I.
Siberian Federal University, Krasnoyarsk, Russia

The method of inductive assertions was originally formulated for the verification of successive imperative programs. The use of this method for the verification of functional dataflow parallel programs developed in the Pythagor programming language is suggested in this paper. The adaptation of the inductive assertions method presented in the paper is based on the dataflow graph of a functional dataflow model of parallel calculation. A user's specification is specified by classical initial and intermediate assertions described by special formulas.. These formulas are implemented by using a special language.

An initial assertion describes a general form of an argument of a function. To describe it there are constants that determine the type of the argument. The formulas can specify the intervals in which arguments can be changed. Intermediate assertions correspond to the nodes of an information graph of the program. All basic language statements of the Pythagor language and formulas used to the initial assertion are available to describe intermediate assertions. This makes it possible to impose various conditions and constraints on the program which is running. Conditions must be formulated in such a way that they return a Boolean value. In this case it possible to make a definite conclusion whether the execution of the program corresponds to the specified assertion or not, In addition, the obtained Boolean value is used for displaying errors occurring in the program. These errors are displayed in the information graph of the program. The application of the method of inductive assertions to verify functional dataflow parallel programs makes it possible to check the agreement of the program computations with a user’s specification and to estimate the results of the program computation by intervals. This simplifies the process of developing, testing and debugging of functional dataflow parallel programs.

Keywords: verification, specification, correctness, functional dataflow parallel programming.

REFERENCES

1. Legalov A.I. Funktsional'nyi yazyk dlya sozdaniya arkhitekturno-nezavisimykh pa-rallel'nykh programm [Functional language for creating of architectural independent parallel programs]. *Vychislitel'nye tekhnologii – Computational Technologies*, 2005, vol. 10, no. 1, pp. 71-89.
2. Udalova Yu.V., Legalov A.I., Sirotinina N.Yu. Sredstva otladki funktsional'no-potokovykh parallel'nykh programm [Environment for debugging of functional and dataflow parallel programs]. *Doklady Akademii Nauk Vysshei Shkoly Rossiiskoi Federatsii – Proceedings of the Russian Higher School Academy of Sciences*, 2008, no. 1 (10), pp. 96-105.
3. Udalova Yu.V., Legalov A.I., Sirotinina N.Yu. Metody otladki i verifikatsii funktsional'no-potokovykh parallel'nykh programm [Debug and Verification of Function-Stream Parallel Programs]. *Zhurnal Sibirskogo federal'nogo universiteta. Seriya: Tekhnika i tekhnologii – Journal of Siberian Federal University. Engineering & Technologies*, 2011, vol. 4, no. 2, pp. 213-224.
4. Liskov B., Guttag J. *Abstraction and specification in program development.* Massachusetts, MIT Press., 1986. 488 p. (Russ. ed.: Liskov B., Gateg Dzh. *Ispol'zovanie abstraktsii i spetsifikatsii pri razrabotke programm*. Moscow, Mir Publ., 1989. 424 p.).

СВЕДЕНИЯ ОБ АВТОРАХ

Легалов Александр Иванович – родился в 1956 году, д-р техн. наук, профессор, заведующий кафедрой вычислительной техники, Сибирский федеральный университет. Область научных интересов: технологии разработки программного обеспечения, языки программирования, параллелизм. Опубликовано 160 научных работ. (Адрес: 660074, Россия, Красноярск, ул. акад. Киренского, 26. Email: legalov@mail.ru).

Legalov Alexander Ivanovich(b. 1956) – D.Sc.(Eng.), professor, Computer Engineering Department, Siberian Federal University. His research interests are currently focused on software engineering, programming languages, and parallelism. He is author of 160 scientific papers. (Address: 26, Kirensky Street, Krasnoyarsk, 660074, Russia. Email: legalov@mail.ru).

Удалова Юлия Васильевна – родилась в 1982 году, старший преподаватель кафедры «Высокопроизводительные вычисления», Сибирский федеральный университет. Область научных интересов: параллельное программирование, методы верификации программ. Опубликовано 20 научных работ. (Адрес: 660074, Россия, Красноярск, ул. акад. Киренского, 26. Email: uuuu82@inbox.ru).

Udalova Julia Vasilevna(b. 1982) – senior teacher of High Performance Computing Department, Siberian Federal University. Her research interests are currently focused on parallel programming and methods of program verification. She is author of 20 scientific papers. (Address: 26, KirenskyStreet, Krasnoyarsk, 660074, Russia. Email:JUdalova@sfu-kras.ru,uuuu82@inbox.ru).

Статья поступила 18 мая 2014 г.
Received May 18, 2014

To Reference:

Udalova Yu.V., Legalov A.I. Verifikatsiya funktsional'no-potokovykh parallel'nykh programm metodom induktivnykh utverzhdenii [Verification of functional dataflow parallel programs using the method of inductive assertions]. *Doklady Akademii Nauk Vysshei Shkoly Rossiiskoi Federatsii – Proceedings of the Russian Higher School Academy of Sciences*, 2014, no. 2-3 (23-24), pp. 125-132.