УДК 004.052.42

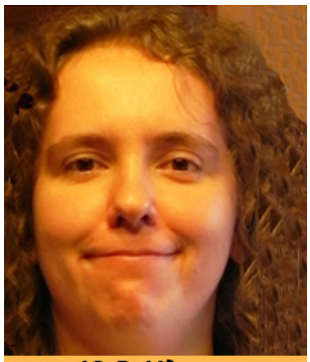
ВЕРИФИКАЦИЯ ФУНКЦИОНАЛЬНО-ПОТОКОВЫХ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ С ПОМОЩЬЮ ИНТЕРВАЛЬНЫХ ФОРМУЛ

***Юлия Васильевна Удалова**, к.т.н., доцент*
Тел.: 8 908 210 7282, e-mail: uuuu82@inbox.ru
Сибирский федеральный университет
http://www.sfu-kras.ru

В статье описан оригинальный метод верификации программ на функционально-потоковом языке параллельного программирования Пифагор, использующий для спецификации программы и представления промежуточных утверждений формулы с интервальными константами.

Ключевые слова: верификация, спецификация, параллельное программирование, функциональное программирование.

Современное параллельное программирование в основном опирается на императивную парадигму. Вместе с тем развиваются и другие альтернативные подходы, например, функционально-потоковое параллельное (ФПП) программирование [1], ориентированное на разработку архитектурно-независимых параллельных программ. ФПП программа поддерживает параллелизм на уровне операций, задержанные вычисления, может быть представлена в виде ациклического информационного графа. Множество реализованных функционально-потоковых языков параллельного программирования крайне мало, один из его представителей – язык Пифагор [2], являющийся научной разработкой авторского коллектива Сибирского федерального университета. Задача верификации [3] в настоящее время является актуальной и для параллельного программирования вообще, и тем более, для развивающегося направления ФПП программирования, специальные особенности модели которого требуют учета при проведении верификации.


Ю.В. Удалова

В статье предлагается метод верификации ФПП программ на языке Пифагор, основанный на адаптации классических методов индуктивных утверждений и индукции [3] к ФПП модели, оперирующий спецификацией и выведенными утверждениями, заданными через интервальные формулы.

Верификация функционально-потоковых параллельных программ на языке Пифагор с помощью интервальных формул

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 [3], изначально сформулированные для последовательных императивных программ, к которым относится выбранный метод индуктивных утверждений. Применение метода индуктивных утверждений базируется на знании порядка обхода операторов программы. Для ФПП программы обход операторов производится в соответствии с информационными зависимостями между вершинами-операторами информационного графа программы [4].

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 интервальных 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 (1, ~unknownbool, ~gt 0), т.е. аргумент функции является списком данных, первый элемент которого равен единице, 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 как вышеописанные интервальные формулы, так 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 операторов назначаются автоматически перед началом верификации и видны пользователю).

Представленный метод верификации автоматизирован в программной среде для разработки, отладки и верификации программ на языке Пифагор [5]. Автоматизированы представление информационных графов функций пользователю, описание начального утверждения, расстановка промежуточных утверждений на графе, автоматизирован и сам процесс верификации, т.е. вывод утверждений при обходе операторов на информационном графе, при переходе на последующие итерации рекурсий и подсвечивание на графе некорректных, т.е. не соответствующих пользовательским промежуточным утверждениям, вычисленных значений.

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 промежуточных утверждений, но рекомендуется выстраивать их так, чтобы они возвращали булевы значения, – это позволит при прохождении процесса верификации отмечать на графе корректные и некорректные операторы.

Процесс верификации опирается на базу правил, определяющих срабатывание операторов языка Пифагор над интервальными формулами, при составлении базы правил были использованы формулы интервального анализа [6]. Сам процесс верификации 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.

Если верифицируется рекурсивная функция, то автоматический верификатор рассмотрит столько ее итераций, сколько будет нужно пользователю, либо достигнет выхода из рекурсии раньше. При этом на каждой итерации будут отмечаться корректные и некорректные в смысле соответствия промежуточным утверждениям спецификации операторы. Здесь возможна ситуация, когда один и тот же оператор на различных итерациях рекурсии будет являться то корректным, то некорректным, анализ подобной ситуации также ложится на пользователя, проводящего автоматическую верификацию. Это же относится и к произведению выводов об общих особенностях работы рекурсивной функции на основе детальной информации о прохождении ее первых N итераций.

Возможно проведение верификации только при задании начального утверждения (т.е. общего вида аргумента функции) и при отсутствии промежуточных утверждений. В этом случае польза верификации состоит в получении интервальных оценок результатов срабатывания операторов программы и вычисляемых функциями результатов. Интервальная оценка кроме математического анализа производимых программой вычислений имеет и алгоритмический смысл – это выход значений операторов за границы типов данных, что приведет к ошибке при выполнении программы.

Пример спецификации и верификации программы на языке Пифагор

Функция Abs получает число P и вычисляет его модуль.

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

Информационный граф функции Abs и этот же граф с утверждениями, полученными после спецификации и верификации, представлены на рисунке ниже.

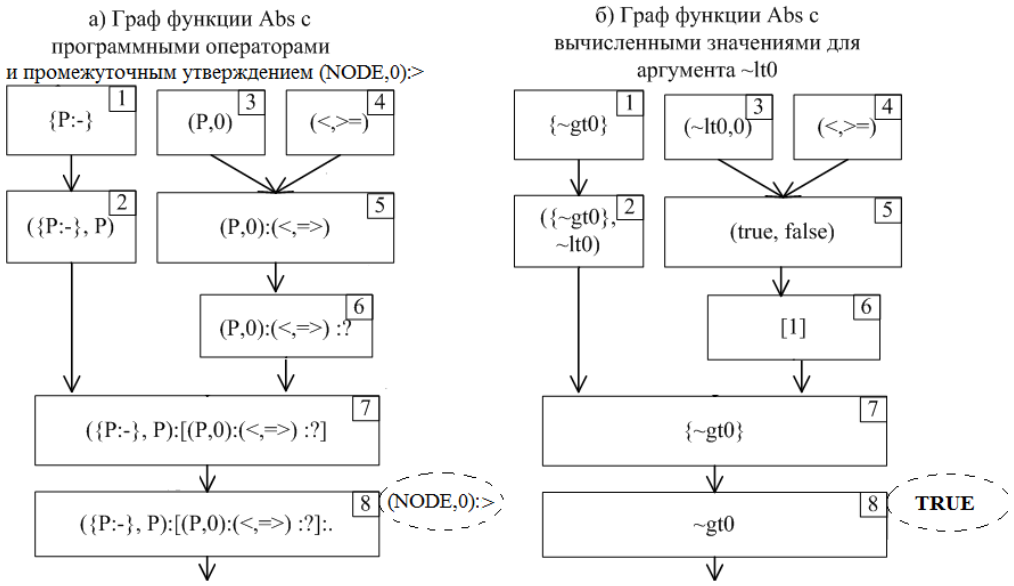


Рисунок. Информационные графы функции Abs

Начальное утверждение спецификации здесь задано как ~lt 0, т.е. аргумент функции P – это число меньше нуля. К результирующему оператору функции под номером 8 пользователем задано промежуточное утверждение (требование) (NODE, 0):>, т.е. результат оператора №8 – это число больше нуля. Автоматически произведенная верификация установила, что программа корректна относительно заданной спецификации, т.е. функция Abs для любого аргумента меньше нуля выдаст результат больший нуля, что

подтверждается полученным результатом TRUE (истина) для промежуточного утверждения и корректностью базы правил работы операторов языка Пифагор над интервальными данными.

Рассмотренная функция Abs проста, но иллюстрирует процесс автоматической верификации, по аналогичным принципам он будет проходить для более сложных функций, а для рекурсивных функций рассмотрит несколько их начальных итераций, предоставляя на каждой итерации граф с новыми вычисленными значениями операторов и промежуточных пользовательских утверждений.

Заключение

В данной работе новыми являются следующие положения и результаты:

– адаптация метода индуктивных утверждений, изначально сформулированного для императивных последовательных программ, к функционально-потоковым параллельным программам, а конкретно к информационным графам их функций;

– спецификация функционально-потоковых параллельных программ с помощью интервальных формул;

– оригинальные формулы спецификации для функционально-потокового языка параллельного программирования Пифагор;

– оригинальная программная среда для разработки и автоматической верификации программ на функционально-потоковом языке параллельного программирования Пифагор.

Представленный метод верификации:

– обеспечивает автоматическую верификацию программы на языке Пифагор над обобщенным множеством входных данных, описанным через интервальные формулы;

– позволяет установить соответствие вычислений, выполняемых программой, утверждениям спецификации пользователя (т.е. другими словами позволяет подтвердить или опровергнуть корректность программы);

– предоставляет интервальные оценки вычисляемых программными операторами значений.

Литература

1. *Легалов А.И., Савченко Г.В., Васильев В.С.* Событийная модель вычислений, поддерживающая выполнение функционально-потоковых параллельных программ // Системы. Методы. Технологии. 2012. № 1 (13). С. 113-119.

2. *Легалов А.И., Привалихин Д.В.* Особенности функционального языка параллельного программирования «Пифагор» // Высокопроизводительные вычисления на кластерных системах: материалы четвертого Международного научно-практического семинара и Всероссийской молодежной школы. Самара, 2004. С. 173-179.

3. *Лисков Б., Гатэг Дж.* Использование абстракций и спецификаций при разработке программ. – М.: Мир, 1989. 424 с.

4. *Редькин А.В., Легалов А.И.* Событийное управление выполнением функционально-потоковых параллельных программ // Научный вестник НГТУ. 2008. № 3 (32). С. 111-120.

5. *Удалова Ю.В., Легалов А.И., Сиротинина Н.Ю.* Отладка программ на функционально-потоковом параллельном языке Пифагор с подстановкой интервальных значений // Ползуновский вестник. 2013.№ 2. С. 46-48.

6. *Алефельд Г., Майер Г.* Интервальный анализ: теория и приложенияю. URL: http://www.sbras.ru/interval/Introduction/ISurveyRus.pdf.

Verification of parallel functional dataflow programs using interval formulas

***Udalova Julia Vasilievna**, PhD, Docent*

The article describes the original method of verification of programs in functional-stream parallel programming language Pythagoras using interval constants for the specifications of the program.

Keywords – verification, specification, parallel programming, functional programming.