УДК 004.432.42

М. С. Кропачева
Сибирский федеральный университет,
г. Красноярск, Красноярский край, Россия

ФОРМАЛЬНАЯ ВЕРИФИКАЦИЯ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

Рассматривается формальная верификация функционально-потоковых параллельных программ. Ошибки, характерные для императивных параллельных программ, отсутствуют в функционально-потоковых параллельных программах, что позволяет упростить верификацию. Анализируется корректность примера программы на языке Пифагор.

Ключевые слова: функционально-потоковое параллельное программирование, формальная верификация, язык программирования Пифагор.

M. S. Kropacheva
Siberian Federal University, Krasnoyarsk, Russia

FORMAL VERIFICATION OF PARALLEL PROGRAMS

Formal verification of functional dataflow parallel programs is considered. Errors that are typical for imperative parallel programs are entirely absent in functional dataflow parallel programs. As a result, formal verification of such programs is easier. The correctness of an example program in Pifagor language is analyzed.

Key words: functional dataflow parallel programming, formal verification, Pifagor programming language.

Современная тенденция в развитии высокопроизводительных вычислений приводит к появлению новых ошибок в написанных в традиционном императивном стиле параллельных программах. Напротив, разработка программ на функционально-потоковых языках параллельного программирования позволяет не только упростить разработку параллельных программ, но и использовать для их верификации формальные методы.

Под формальной верификацией понимается формальное доказательство корректности программы, которое заключается в установлении соответствия между программой и требованиями к программе, описывающими цель разработки [1].

В настоящее время существуют различные подходы к формальной верификации программ. Основными являются метод проверки моделей и дедуктивный анализ. Метод проверки моделей позволяет осуществить автоматизируемый перебор всех возможных вариантов выполнения программы, но применим только в случае, когда программа принимает конечное число состояний [2]. Этот метод достаточно эффективно используется при анализе взаимодействия параллельно функционирующих процессов.

Другой подход к формальной верификации программ направлен на анализ логики преобразования данных, определяемой решением поставленной задачи. Обычно такие задачи имеют бесконечное число состояний, и их корректность проверяется с помощью дедуктивного анализа. Основоположниками данного метода являются Хоар и Флойд [3, 4]. Дедуктивный анализ – наиболее универсальный метод формальной верификации. Основная идея метода заключается в том, что последовательными преобразованиями программа переводится в

формулу логики, а требования или спецификация к программе либо изначально формулируются на языке логики, либо переводятся с какого-либо языка на язык логики. После этого доказательство корректности программы сводится к доказательству эквивалентности двух формул, что осуществляется с помощью методов, разработанных в логике. Данный подход хорошо проработан и частично автоматизирован для последовательных программ. Однако процесс доказательства сильно усложняется для параллельных императивных программ.

Если считать, что корректная программа должна, проработав, завершиться и вернуть в качестве результата ответ (программы, для которых предполагается бесконечная работа в цикле, не рассматриваются), то ошибки последовательных императивных программ можно разделить на две группы:

1) ошибки, в результате которых программа не завершается корректно; они могут быть вызваны входом в бесконечный цикл или вызовом частично определенных функций со значением аргумента вне области определения (например, деление на ноль), приводящим к аварийного выходу из программы;

2) семантические ошибки, при которых программа в случае завершения возвращает неверный результат.

Помимо этих ошибок параллелизм добавляет ошибки ещё двух типов, выявить которые намного сложнее:

1) взаимная блокировка – ситуация, при которой несколько процессов находятся в состоянии бесконечного ожидания ресурсов, захваченных самими этими процессами, что приводит к зависанию или аварийному завершению программы;

2) состояние гонки – ошибка, при которой работа системы зависит от того, в каком порядке выполняются части кода, в частности, подобная ошибка возникает, когда несколько процессов неконтролируемо обращаются к одной и той же разделяемой переменной, при этом значение переменной зависит от порядка, в котором процессы получают к ней доступ.

В обоих случаях причина ошибок – конфликты и неправильное использование ресурсов. В предложенной А. И. Легаловым модели функционально-потоковых параллельных вычислений и языке программирования Пифагор, реализующем эту модель, подобные ошибки отсутствуют [5]. В основе модели лежит управление вычислениями по готовности данных. Вычисления протекают внутри бесконечных ресурсов. В языке отсутствуют переменные (используется принцип единственного присваивания), что позволяет избежать конфликтов, связанных с совместным использованием памяти параллельными процессами. В языке нет операторов цикла, и все циклические конструкции реализуются через рекурсии. Специфика модели позволяет представлять отношения между функциями языка в виде дерева (ациклического информационного графа), у которого функции из разных ветвей могут выполняться параллельно. При таком подходе есть возможность описать максимальный параллелизм, ограниченный только информационными связями, присущими конкретной задаче. В результате написанную на функционально-потоковом языке параллельного программирования программу можно перенести на любую архитектуру, и перенос сводится к распределению ресурсов в соответствии с архитектурой.

Таким образом, анализ корректности программ на функционально-потоковом языке параллельного программирования в целом сводится к анализу ошибок, аналогичных последовательному языку программирования.

Открытым остаётся вопрос о переносе существующих методов формального доказательства корректности императивных программ на функционально-потоковые параллельные программы. Поэтому актуальной является разработка формальных методов верификации для функционально-потоковых языков параллельного программирования, обеспечивающих проверку логики функционирования программ.

Рассмотрим формальное доказательство корректности простой функционально-потоковой параллельной программы, вычисляющей факториал числа. Код программы на языке Пифагор в виде информационного графа приведён на рис. 1. Программа представляет собой рекурсивную функцию fact, которая принимает в качестве входного аргумента число x. В тёмных кружках на рисунке указаны идентификаторы к соответствующим дугам.

Сформулируем для функции fact следующую спецификацию на естественном языке: «Если входное значение x – целое число больше нуля, и произведение чисел от 1 до x не превышает максимального целого INT_MAX, которое допускает тип int, то после выполнения функция fact возвращает результат перемножения чисел от 1 до x, если $x > 0$ или 1, если $x = 0$». В виде формул логики исчисления

предикатов первого порядка, расширенной арифметикой, спецификацию функции fact можно представить в виде двух формул:

$$(x \in \text{int}) \wedge (x \geq 0) \wedge \left(\prod_{i=1}^{x} i \leq \text{INT\_MAX} \right), \quad (1)$$

$$\left(\left(r = \prod_{i=1}^{x} i \right) \wedge (x > 0) \right) \vee \left((r = 1) \wedge (x = 0) \right), \quad (2)$$

где r – результат работы программы. Первая из формул описывает ограничения на входной аргумент программы, вторая – требования к результату работы программы. Для доказательства корректности функции fact необходимо показать, что если входной аргумент x удовлетворяет ограничениям (1), то результат вычислений всегда будет удовлетворять формуле (2).

Рассмотрим процесс вычисления функции fact. При $x = 0$ или 1 должно быть возвращено значение 1, при $x > 1$ должен быть совершен рекурсивный вызов fact с аргументом $x - 1$. Проверка истинности одного из условий $x <= 1$ или $x > 1$ выполняется в правой ветви, содержащей функцию «?». Параллельно в левой ветви формируется список p из двух элементов: «1» и «задержанный список» (на рисунке он выделен пунктиром), содержащий подграф с рекурсивным вызовом функции fact. Операторы задержанного списка выполняются только после снятия задержки, даже если все аргументы готовы. Выбор одного из элементов в списке p осуществляется в зависимости от того, какое из условий $x <= 1$ или $x > 1$ верно. Во втором случае выбирается задержанный список, с него снимается задержка функцией «.», и происходит рекурсивный вызов. В случае $x <= 1$ из списка p выбирается 1, а вычисление функций задержанного списка вообще не происходит.

Для того чтобы доказать, что рекурсивная функция завершается и корректна, введём ограничивающую функцию. Ограничивающая функция – это ограниченная снизу функция, переводящая аргумент рекурсивной функции во множество натуральных чисел, при этом все аргументы, для которых выполняется условие завершения, отображаются в единицу. С помощью этой функции корректность рекурсии доказывается по индукции.

Зададим ограничивающую функцию для fact:

$$f(x) = \begin{cases} 1, & x = 0; \\ x, & x > 0. \end{cases}$$

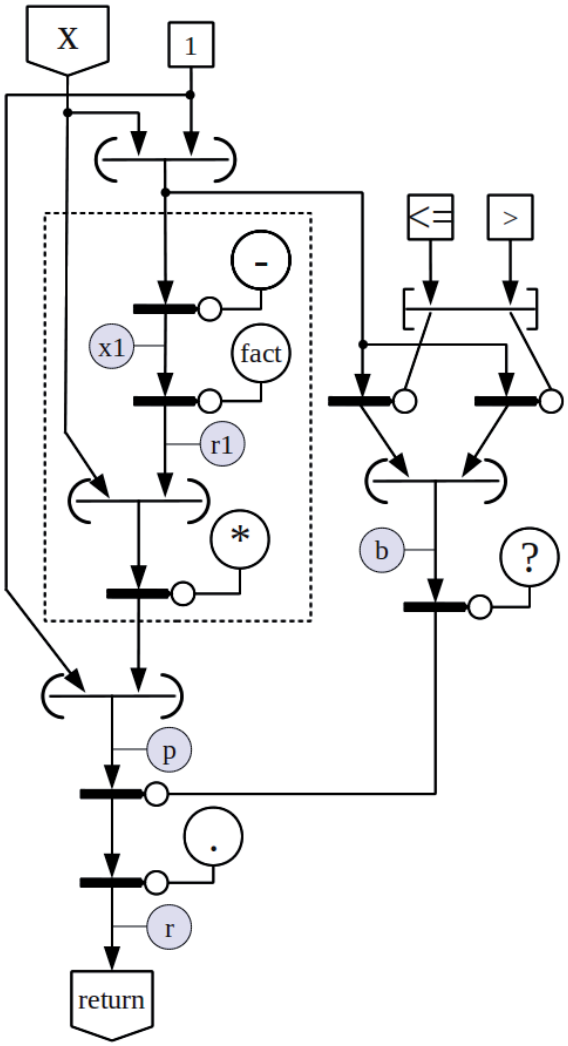

Рис. 1. Информационный граф программы

Докажем корректность функции fact с помощью индукции по значениям ограничивающей функции.

База индукции. Функция f принимает значение, равное единице, если $x = 1$ или $x = 0$. Эти значения удовлетворяют формуле (1). Для всех остальных значений x, удовлетворяющих формуле (1) значение функции f больше единицы.

Если функция fact вызвана с аргументом $x = 1$ или $x = 0$, то готовность значения x запускает вычисления. Аргумент x вместе с константой 1 формирует список из двух элементов (x, 1). Как только список сформирован, начинают выполняться функции «<=» и «>». Эти функции находятся в разных ветвях информационного графа программы, поэтому могут выполняться параллельно. Согласно семантике данных функций они возвращают булевскую константу true или false, в зависимости от значения своего аргумента. В рассматриваемом случае $x = 0$ или $x = 1$, поэтому для

аргумента (x, 1) функция «<=» вернёт значение true, а функция «>» – значение false. Эти константы формируют список $b \equiv$ (true, false), который является входным аргументом функции «?». Согласно своей семантике функция «?» возвращает порядковый номер того элемента входного списка, чьё значение равно true, поэтому для аргумента (true, false) функция «?» возвращает 1. Полученный номер используется как функция, выбирающая первый элемент из списка p, равный 1. Единица передаётся функции «.», которая не оказывает никакого воздействия на аргумент, и функция fact возвращает значение 1. Таким образом, функция fact завершается и возвращает результат, который, очевидно, удовлетворяет формуле (2).

Шаг индукции. Возьмём аргумент $x > 0$, удовлетворяющий формуле (1), пусть значение ограничивающей функции $f(x)$ равно N. Предположим, что функция корректна для всех аргументов рекурсивного вызова, у которых значение ограничивающей функции меньше N.

Вычисление функции fact для $x > 0$ начинается аналогично случаю, когда $x = 1$ или $x = 0$. Однако функция «<=» возвращает значение false, а функция «>» – значение true, поэтому $b \equiv$ (false, true), функция «?» возвращает 2, и из списка p выбирается второй аргумент, который является задержанным списком. Этот задержанный список передаётся функции «.», которая снимает задержку, и в результате начинает выполняться подграф, содержащийся в задержанном списке. Функция минус «–» получает аргумент (x, 1), и, согласно своей семантике, возвращает целочисленное значение $x1 \equiv (x-1)$, которое поступает на вход рекурсивному вызову функции fact. Аргумент $x1$ является входным для функции fact, поэтому должен удовлетворять её спецификации, иначе функция не будет корректна, так как в ней будет некорректный рекурсивный вызов самой себя. Необходимо показать, что $x1$ удовлетворяет формуле (1), если в выражении заменить x на $x1$:

$$(x_1 \in \text{int}) \wedge (x_1 \geq 0) \wedge \left(\prod_{i=1}^{x_1} i \leq INT\_MAX \right),$$

очевидно, что формула истинна, если $x1 \equiv (x-1)$, выполнено условие (1) и $(x > 1)$.

Покажем, что значение ограничивающей функции аргумента $x1$ всегда меньше N:

$$f(x_1) = f(x-1) = f(N-1) = N-1 < N.$$

Тогда по индуктивному предположению результат рекурсивного вызова функции $r1$ будет удовлетворять выражению (2), если переменную r заменить на $r1$, а x на $x1$:

$$\left(\left(r_1 = \prod_{i=1}^{x_1} i \right) \wedge (x_1 > 0) \right) \vee \left((r_1 = 1) \wedge (x_1 = 0) \right). \quad (3)$$

При дальнейшем выполнении функции элементы $r1$ и x формируют список (x, $r1$), который передаётся функции умножения «*». По семантическим правилам функция умножения возвращает целочисленное значение r, равное произведению своих аргументов, то есть $r \equiv (x*r1)$. Значение r будет результатом работы функции fact.

Покажем, что r удовлетворяет выражению (2). Из истинности выражения (3) имеем, что $r1$ равняется результату перемножения чисел от 1 до $x1$, тогда r будет равняться результату перемножения чисел от 1 до x. Таким образом, доказано, что функция fact удовлетворяет своей спецификации.

Благодаря тому, что язык программирования Пифагор не ограничивает параллелизм разрабатываемых на нём программ, он может быть использован в качестве инструмента для обобщённой спецификации параллельных программ, обеспечивая их более простую формальную верификацию, тестирование и отладку, а затем – формальное преобразование (автоматическое или ручное) программы в представление, предназначенное для выполнения на конкретной архитектуре.

В дальнейшем предполагается построить аксиоматическую теорию, которая позволит частично автоматизировать процесс верификации.

Библиографические ссылки

1. Непомнящий В. А., Рякин О. М. Прикладные методы верификации программ. – М. : Радио и связь, 1988. – 255 с.
2. Кларк М., Грамбер, О., Пелед Д. Верификация моделей программ: Model Checking / под ред. Р. Смелянского. – М. : МЦНМО, 2002. – 416 с.
3. Hoare C. A. Retrospective: An axiomatic basis for computer programming // Communications of the ACM. – 1969. – Vol. 10. – № 12. – P. 576–585.
4. Floyd R. W. Assigning meaning to programs // Proc. of Symposium in Applied Mathematics. J. T. Schwartz, ed. Mathematical Aspects of Computer Science. – 1967. – № 19. – P. 19–32.
5. Легалов А. И. Функциональный язык для создания архитектурно-независимых параллельных программ // Вычислительные технологии. – 2005. – № 1 (10). – С. 71–89.