УДК 004.052.42

ПОДХОД К РЕШЕНИЮ ПРОБЛЕМ ВЕРИФИКАЦИИ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

О. С. Крюков

Тульский государственный университет, пр. Ленина, 92, 300012, г. Тула, Россия,
ol_kryukov97@mail.ru

Рассматривается проблема верификации параллельных программ. Приводится анализ основных методов верификации, используемых для проверки корректности программного кода. Формулируются основные требования к верификатору параллельных программ.

Ключевые слова: верификация, параллельность, формальная верификация, проверка эквивалентности, сети Петри.

APPROACH TO SOLVING PROBLEMS OF PARALLEL PROGRAMS VERIFICATION

O. S. Kryukov

Tula State University, ave. Lenin, 92, 300012, Tula, Russia,
ol_kryukov97@mail.ru

The problem of parallel programs verification is considered. An analysis of the main verification methods used to check the correctness of the program code is given. The main requirements for the verifier of parallel programs are formulated.

Keywords: verification, parallelism, formal verification, equivalence check, Petri nets.

Ключевой особенностью современных вычислительных систем является многопроцессорная и/или многоядерная архитектура. Подобные системы способны предоставлять кратное увеличение производительности при выполнении программного кода по сравнению с последовательными вариантами. В связи с этим, все больше современного программного обеспечения разрабатывается с учетом возможного параллелизма для наиболее эффективного использования вычислительных мощностей.

Однако, компьютерные программы зачастую содержат в себе ошибки, и усложнение технологий приводит к возрастанию их количества, а одновременный рост количества и критичности выполняемых им функций влечет рост ущерба от этих ошибок. Стоит отметить, что определенные системы уже отличается таким уровнем сложности, что люди в принципе не могут избежать ошибок, а быстрое их обнаружение затруднено. Для обеспечения корректности и надежности работы таких систем большое значение имеют различные методы верификации и валидации, позволяющие выявлять ошибки на разных этапах разработки и сопровождения ПО, чтобы последовательно устранять их.

При работе с параллельной системой необходимо помнить о возможном непредсказуемом поведении программного кода, связанного с конкуренцией процессов, гонками процессов или ошибками доступа к памяти. «Плавающий» характер таких ошибок отражается и на способе их обнаружения: многочисленные пробные запуски с целью отследить их возникновение. Подобное решение не очень удобно и может быть крайне затруднено в случае работы с крупной информационной системой или системами, требующими большого объема вводимых вручную данных, все комбинации которых проверить невозможно.

В связи с этим применяются различные системы, автоматизирующие данный процесс. В настоящее время существует множество различных методов верификации, среди которых можно выделить статических анализ и динамические методы [1] как одни из наиболее популярных решений, применяемых автоматизированно.

Статический анализ позволяет проверять корректность выполнения правил, принятых для построения артефактов жизненного цикла программного обеспечения [2; 3]. Ключевой особенностью подобных верификаторов является то, что проверка корректности программного кода выполняется без его непосредственного запуска на вычислительной машине, при этом проверяются даже редко посещаемые в процессе выполнения участки. Результатом работы статического анализатора является список потенциальных проблемных мест, реальную важность которых может оценить только квалифицированный специалист. Еще одной серьезной проблемой статического анализа является его малая применимость в диагностике утечек памяти и ошибок параллельных реализаций, так как данные виды ошибок не связаны с корректностью кода.

Динамический анализ [2] представляет собой анализ программы непосредственно при его выполнении, а решение о корректности принимается на основе полученных результатов. Таким образом можно оценивать потребление системой ресурсов, выявляя утечки памяти, и определять состояния гонки. Однако, как было сказано ранее, верификация на основе анализа выполнения программы не является достаточно надёжным решением, что относится и к динамическим верификаторам. Таким образом, к недостаткам динамической верификации

можно отнести отсутствие гарантии полного покрытия программного кода и серьезные затраты процессорного времени и ресурсов памяти на анализ.

Обеспечить полное покрытие исходного кода без необходимости его выполнения могут верификаторы, основанные на формальных методах верификации. Формальная верификация, в контексте аппаратных и программных систем, является способом доказать или опровергнуть правильность алгоритмов, лежащих в основе системы по отношению к определенной формальной спецификации или свойствам, используя математические методы.

Формальная верификация может выполняться с применением как логико-алгебраических моделей, так и исполнимых. Однако, в связи с необходимостью структурного контроля последовательности операций программного кода, исполнимые модели являются более предпочтительными. Кроме того, исполнимые модели предоставляют более наглядную информацию о процессах, выполняемых в программе. К формальной верификации на основе исполнимых моделей относятся методы проверки моделей: проверка эквивалентности и проверка выполнимости. Проверка моделей или проверка свойства представляет собой способ верификации, проверяющий соответствует ли модель заданной спецификации.

Для формальной верификации по методу проверки эквивалентности необходимо наличие корректного последовательного алгоритма, представляющего собой основу параллельной программы. Его корректность аналогично может быть верифицирована. Непосредственно верификация параллельной программы заключаются в анализе эквивалентности моделей последовательной и параллельной программ.

Верификация по методу проверки выполнимости заключается в проверке выполнимости моделью спецификаций, как правило описываемых на языке темпоральной логики. Минусом данного метода является то, что требования к описанию спецификаций, которым должна удовлетворять параллельная программа, требуют дополнительной проверки корректности и полноты описания [4].

Таким образом, верификация по методу проверки эквивалентности представляется лучшим вариантом для проверки корректности параллельных программ, так как не требует написания отдельных спецификаций и выполнения их проверки.

В качестве исполнимой модели, применяемой при верификации, могут выступать различные конечные автоматы, системы помеченных переходов, сети Петри и т. п. Выбираемая модель должна позволять осуществлять контроль порядка выполнения и последовательности доступа к памяти, а, следовательно, должна отображать как поток управления, так и информационные зависимости между операциями. Данным требованиям в полной мере удовлетворяют сети Петри, дополненные семантическими связями [4–6].

В соответствии с проведенным анализом можно представить следующие требования к верификатору параллельной системы:

1) анализ без непосредственного запуска анализируемого программного кода;
2) полное покрытие анализируемого программного кода;
3) контроль последовательности доступа к памяти;
4) контроль порядка выполнения операций.

Выполнение данных требований позволит реализовать верификатор, способный осуществлять полноценную автоматизированную проверку корректности параллельной программы без необходимости многократного запуска.

Библиографический список

1. Кулямин В. В. Методы верификации программного обеспечения // Конкурс обзорно-аналитических статей по направлению «Информационно-телекоммуникационные системы». 2008.
2. Гурин Р. Е., Рудаков И. В., Ребриков А. В. Методы верификации программного обеспечения // Машиностроение и компьютерные технологии. 2015. № 10. С. 235–250.
3. Мордань В. О. Комбинация методов статической верификации композиции требований // Труды ИСП РАН. 2017. № 3. С. 151–170.
4. Ивутин А. Н., Трошина А. Г. Метод формальной верификации параллельных программ с использованием сетей Петри // Вестник РГРТУ. 2019. № 70. С. 15–26.
5. Ivutin A. N., Voloshko A. G. Method of Formal Verification of Program Code based on Petri Net with Additional Semantic Relations. 2020 ELEKTRO. 2020. Pp. 1–6. DOI: 10.1109/ELEKTRO49696.2020.9130267.
6. Ivutin A. N., Voloshko A. G., Izotov V. N. Approach to Data Race Detection Based on Petri Nets with Additional Semantic Relations. 2020 ELEKTRO. 2020. Pp. 1–5. DOI: 10.1109/ELEKTRO49696.2020.9130252.

Исследование выполнено при финансовой поддержке РФФИ и Правительства Тульской области в рамках научного проекта 19-41-710003 р_а.