ИСПОЛЬЗОВАНИЕ МОДЕЛЕЙ ПРИ ТЕСТИРОВАНИИ МНОГОПОТОЧНЫХ ПРОГРАММ

Буздалов Денис Викторович
ИСП РАН
+7-906-0490038
buzdalov@ispras.ru

Введение

Многопоточные программы сложны по своему поведению, потому что то, как они себя ведёт зависит не только от исполнимого кода и внешних данных, с которыми программа работает. Поведение такой программы сильно зависит от окружения, в котором она выполняется. Как следствие, при написании многопоточной программы легко допустить ошибку, так как программист может не учесть некоторые варианты влияния окружения на поведение своей программы.

Но такие программы часто используются там, где от корректности их поведения зависит жизнь и здоровье людей, либо там, где ошибки в программах могут повлечь большие финансовые затраты. Поэтому проверка корректности таких программ является очень важной задачей. Таким образом, встаёт задача проверки корректности таких программ.

Существует несколько подходов к решению этой задачи [1]. Они различаются как трудоёмкостью, так и областью своего применения. К примеру, для своей работы они могут требовать дополнительную информацию о программе или среде выполнения, что иногда ограничивает сферу, в которой они применимы. Также, подходы могут различаться тем, как задаётся проверяемая корректность программы.

Среди распространённых подходов можно выделить:

- верификация моделей программ (model-checking)
- тестирование
  - тестирование на основе исходного кода
  - тестирование на основе бинарного представления программы.

Методы проверки корректности, основывающиеся на этих подходах сталкиваются с рядом проблем. Некоторые из них:

- часто сложно определить полноту сделанной проверки
- количество вариантов выполнения многопоточной программы увеличивается экспоненциально в зависимости от количества параллельных потоков выполнения программы и количества инструкций в них.

С последней проблемой сталкиваются все методы, основанные на динамической проверке программы, а не на статическом анализе. Упомянутая выше верификация моделей программ и большинство методов тестирования на основе бинарного кода относятся к таким методам. Остановимся на этой проблеме.

Для её решения существует ряд алгоритмов ограничения множества вариантов работы программы (*выполнений* программы), которые нужно рассмотреть для проверки программы на корректность.

Существующие подходы

На данный момент разработан ряд методов ограничения множества выполнений программы. Вот некоторые из них:

- Метод устойчивых множеств (persistent sets) [2]
- Метод оцепеневших множеств (sleep sets) [2]
- Метод ограничения числа переключений (context switch bounding) [3]

Первые два метода относятся к группе методов редукции частичных порядков. Основной принцип - использование независимости инструкций или групп инструкций. Инструкции независимы когда они *не изменяют возможности друг друга выполниться* и *выполнение их в любом порядке приводит к одному и тому же состоянию*. Если инструкции независимы, то среди выполнений программы, различающихся только порядком независимых инструкций, для обнаружения ошибки достаточно проверить только одно.

Особенностью этих методов является то, что они требуют для своей работы дополнительную информацию о независимости инструкций.

Плюсом этих методов является то, что если среди всех выполнений программы есть ошибка, то методы редукции частичных порядков ограничат множество выполнений таким образом, что хотя бы одно содержащее ошибку выполнение в нём осталось. Как следствие, эти методы могут применяться для *доказательства* наличия ошибок в программе.

Метод ограничения числа переключений состоит в том, чтобы не рассматривать такие выполнения, в которых встречаются больше, чем заданное число переключений контекста программы (переключений между потоками).

Этот метод не требует информации больше, чем есть в дереве выполнений программы, но зато способен выбросить из рассмотрения все выполнения, содержащие ошибку. Поэтому этот метод *нельзя* использовать для доказательства корректности, однако он успешно используется при тестировании позволяя производить тестирование больших программных систем.

Предлагаемый подход

Модели программ успешно применяются при тестировании однопоточных программ [4, 5]. Существует инструменты, позволяющие описывать модели и применяющие данный подход для тестирования. Примером таких инструментов могут служить поддерживающие технологию UniTESK [6] инструменты JavaTESK и CTESK.

Оказалось, что идеи подхода тестирования на основе модели можно применить для ограничения множества выполнений многопоточной программы.

Предположим, проверяемая программа имеет некоторую модель, с помощью которой можно узнавать корректность выполнения программы. Если на базе такой информации мы сможем понять, что какое-либо выполнение не приведёт к ошибке, то во время проверки программы на корректность не будет необходимости проверять его. Тем самым, уменьшатся затраты на проверку корректности программы.

От того, как задана модель, зависит возможность и лёгкость получения информации, необходимой для описанного подхода. В частности, модели программ упомянутого инструмента JavaTESK позволяют получать такую информацию. Модель определяет некоторые события. Такими событиями могут быть передача данных

программе или её ответная реакция. Модель содержит предикаты корректности поведения программы при возникновении событий. Также, она описывает т.н. сценарий, в котором описывается то, каким образом можно воздействовать на проверяемую программу.

Модель, описанная таким образом, позволяет автоматически формулировать утверждения о том, что все выполнения, последовательность событий которых содержит некоторый префикс, являются корректными. Это означает, что проверять все такие выполнения нет смысла, тем самым мы ограничиваем множества рассматриваемых выполнений.

Описанный подход имеет ограничение по применимости. Его можно применять только если на основе модели можем формулировать утверждения, позволяющие не рассматривать выполнения и если проверяемая программа имеет такую модель. Однако, если эти ограничения удовлетворены, то количество работы, необходимой для проверки корректности программы, может быть уменьшено при использовании данного подхода. Плюсом такого подхода является то, что алгоритмы, основанные на данном подходе, не выкидывают некорректные (то есть содержащие ошибку) выполнения из рассмотрения. Поэтому если в программе содержится ошибка, то она будет обнаружена при проверке множества выполнений, ограниченного этим алгоритмом.

Реализация

Был реализован алгоритм, основанный на описанном подходе. Этот алгоритм использовал модели JavaTESK. Алгоритм был встроен в инструмент проверки многопоточных программ Sapsan и при его работе ограничивал множество рассматриваемых выполнений, наряду с другими алгоритмами.

Сравнивались затраты по времени и памяти, необходимые для проверки программы, с данным алгоритмом и без. Выигрыш по затратам зависел от проверяемой программы. Экономия ресурсов для проверки программы на корректность составила от 30% до 80%.

Заключение

Был применён подход тестирования на основе моделей к проблеме ограничения множества выполнений многопоточной программы. Этот метод применим только если проверяемая программа имеет

модель, обладающую определёнными свойствами. Однако, применение этого метода позволяет заметно снизить затраты на проверку корректности программы.

Список литературы

[1] *Э. М. Кларк мл., О. Грамберг, Д. Пелед.* Верификация моделей программ: Model Checking. *МЦНМО, М., 2002*

[2] *Patrice Godefroid.* Partial-Order Methods for the Verification of Concurrent Systems. *Universite de Liege, 1995*

[3] *Madan Musuvathi and Shaz Quadeer.* Iterative Context Bounding for Systematic Testing of Multithreaded Programs. *LDDI'07, ACM, 2007*

[4] *M. Broy, B. Jonsson, J.-P. Katoen, M. Leuker, A. Pretschner.* Model-Based Testing of Reactive Systems. *Advanced Lactures. LNCS 3472, Springer-Verlag, 2005*

[5] *M. Utting and B. Legeard.* Practical Model-Based Testing: A Tools Approach. *Morgan-Kaufmann, 2007*

[6] *В. В. Кулямин, А. К. Петренко, А. С. Косачёв, И. Б. Бурдонов.* Подход UniTESK к разработке тестов. *Программирование 29(6), 2003*