электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

 # 10, октябрь 2015

УДК 519.686.4

Методы проверки на останов программного кода

***Дикопалова Т.С.**, студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Программное обеспечение ЭВМ и информационные технологии»

Научный руководитель: Ломовской И.В., старший преподаватель
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Программное обеспечение ЭВМ и информационные технологии»
irudakov@bmstu.ru

Проверка программы на останов заключается в доказательстве, может ли данная программа быть выполнена за конечное время. Для этого требуется определить, завершаются ли содержащиеся в программе циклы, рекурсивные вызовы и вызовы внешних функций.

Интерес представляет автоматический анализ на завершаемость, не требующий специальной предварительной ручной разметки кода. Поскольку ресурсы вычислительной системы ограничены, для решения данной задачи можно применять и полный перебор состояний системы, но это является крайне неэффективным способом, так как требует больших вычислительных ресурсов.

Среди основных методов анализа на останов можно перечислить следующие:

- анализ графа потока управления (ГПУ);
- методы, основанные на определении отношения порядка;
- метод зависимых пар;
- метод семантических маркировок;
- метод ограниченных систем;
- проверка моделей.

Анализ ГПУ

Часть случаев незавершаемости может быть найдена посредством анализа графа

потока управления программы. Для этого необходимо найти участки кода, которые не завершаются никогда вне зависимости от условий ветвлений. Данная задача сводится к поиску компонент сильной связности на графе. Применить для этого можно любой из известных алгоритмов, например, алгоритм Косарайю или алгоритм Тарьяна. При данном подходе можно найти только явные, никогда не завершаемые блоки кода. Найти же незавершаемые в результате неправильно составленных условий блоки кода не представляется возможным, так как ребро выхода из цикла в графе присутствовать будет, в то время как условие может никогда не выполниться.

Методы, основанные на определении отношения порядка

Проблема останова широко исследовалась с использованием систем переписывания. Классическим методом доказательства завершаемости системы переписывания является определение отношения порядка на ней. При этом может использоваться непосредственное определение правил, что один терм больше другого и т. д., и доказательство, что составленное отношение порядка непротиворечиво. Данные методы применимы как для систем переписывания, так и для непосредственного анализа программ.

Методы нахождения порядка классифицируются по типу отношения порядка, которое используется для доказательства. Среди них можно выделить следующие:

- лексикографический порядок пути [1];
- рекурсивный порядок пути [2];
- полиномиальный порядок [3].

Методы поиска лексикографического порядка пути и рекурсивного порядка пути принимают во внимание позиции термов в правилах: отношение порядка строится на основе того, с левой стороны правила терм находится или с правой. Данные методы неудобно автоматизировать, поэтому чаще используют метод полиномиального порядка.

Широкое распространение получил метод поиска полиномиального порядка. Особенностью его является то, что для доказательства завершаемости необходимо найти некоторое отображение термов на множество рациональных чисел, на котором уже определён естественный порядок. Такие отображения называют ранговыми функциями. Используется данный метод как отдельно, так и в составе других методов, что будет рассмотрено позже.

Для автоматического нахождения линейных ранговых функций был предложен метод,

основанный на представлении программы в виде системы линейных неравенств [3]: линейные неравенства составляются на основе того, как изменяются переменные. На их основе строится задача линейного программирования, которая затем может быть решена, например, симплекс-методом.

Для анализа данным методом программа представляется в виде системы линейных неравенств $(AA')\binom{x}{x'} \leq b$, где x— вектор-столбец переменных в начальном состоянии, x'— вектор-столбец переменных в конечном состоянии (после выполнения), A— коэффициенты при x, A'— матрица коэффициентов при x', b— вектор-столбец свободных коэффициентов. Чтобы программа могла быть представлена в этом виде, она должна отвечать следующим требованиям:

- все переменные должны быть целочисленными;

$$x = (x_{1,}x_{2,} \dots, x_n), x_i \in \mathbb{Z}$$

- переменные должны изменяться линейно;

$$a_1{}'x_1{}' + \cdots + a_n{}'x_n{}' \leq a_1x_1 + \cdots + a_nx_n + a_0, \text{ где} a_i, a'_i \in \mathbb{Z}$$

- условия должны представлять собой линейные неравенства.

$$c_1x_1 + \cdots + c_nx_n \leq c_0, \text{ где} c_i \in \mathbb{Z}$$

Соответственно, $(AA')\binom{x}{x'} \leq b$ представляет собой множество неравенств, представляющих изменения переменных, объединённое с множеством неравенств, представляющих условия циклов.

Для составления задачи линейного программирования вводятся два вектора-столбца переменных: λ_1,λ_2, длина которых равна количеству неравенств в системе линейных уравнений, представляющих программу. Минимизации функция $\lambda_2 b \to min$, ограничения задаются как представлено ниже.

$$\begin{cases} \lambda_{1,}\lambda_2 & \geq & 0 \\ \lambda_1 A' & = & 0 \\ (\lambda_1 - \lambda_2)A & = & 0 \\ \lambda_2(A + A') & = & 0 \\ \lambda_2 b & < & 0 \end{cases}$$

Линейная ранговая функция при этом будет иметь следующий вид.

$$\rho(x) = \begin{cases} \lambda_2 A'x \text{если } \exists x' : (AA')\binom{x}{x'} < b \\ -\lambda_1 b + \lambda_2 b \end{cases}$$

Для доказательства завершаемости может быть достаточно просто определение того,

что у данной системы есть решение, т. е. ранговая функция существует, при этом находить коэффициенты ранговой функции не является необходимы.

Одним из недостатков подобных требования является то, что изменения переменных в рамках одного цикла считаются одновременными. Данное ограничение можно минимизировать посредством предварительной обработки анализируемого кода: построить инварианты циклов и осуществить замыкание переменных [4].

Другим путём расширения класса программ, которые можно проанализировать с помощью данного метода, является разбиение задачи на подзадачи. Если переменная цикла изменяется различно от итерации к итерации, например while(i<10) { if (i > 5) i+=2; else i--; }, то можно рассмотреть отдельно два цикла: где переменная увеличивается и где уменьшается, и затем совместить ограничения [4].

Общим недостатком методов поиска отношения порядка является то, что для доказательства завершаемости необходимо найти отношение порядка, которому удовлетворяли бы все правила системы переписывания, что может быть невозможно, например, когда параметры передаются без изменений. Также недостатком является то, что, поскольку методы разработаны изначально для систем переписывания, они не учитывают конечность типов при исполнении программ на реальных вычислительных машинах.

Метод зависимых пар

Метод зависимых пар заключается в непосредственном применении идеи разбиения задачи доказательства завершаемости некоторого кода на подзадачи и рекурсивное их доказательство. Его достоинством по сравнению с методами поиска отношения порядка является то, что в нём не обязательно искать строгое отношение порядка для всех правил системы [5].

Метод зависимых пар определён для систем переписывания. Для применения его к анализу программ необходимо предварительно преобразовать программу к схожему виду.

Зависимые пары определяются как пары функциональных термов, связанные посредством правил системы переписывания. Критерием завершаемости системы переписывания в данном случае является отсутствие бесконечной цепочки зависимых пар. Для доказательства отсутсвия такой цепочки требуется найти нестрогое отношение порядка для всех правил системы и строгое отношение порядка для всех зависимых пар. Поиск отношения порядка может выполняться с использованием рассмотренных ранее методов.

Метод семантических маркировок

Метод семантических маркировок заключается в анализе не исходной системы переписывания, а эквивалентной ей системы, полученной специальным методом маркировки функциональных термов. Данный метод определён только для систем переписывания. Как утверждает автор метода [6], маркированную систему переписывания зачастую легче анализировать, чем исходную, и существует ряд систем переписывания, для которых не может быть доказана завершаемость напрямую, но может быть доказана для эквивалентной маркированной системы.

Для построения маркированной системы необходимо задать определённым образом функции отображения функциональных термов и термов переменных на некоторую алгебру, а затем задать правила маркировки правил и задания преобразованной системы переписывания. Для доказательства завершаемости преобразованной системы применяется один из ранее описанных методов поиска отношения порядка. Следовательно, данный метод наследует все недостатки применённого вспомогательного метода. Помимо этого, не был предложен алгоритм для автоматического заданий функций отображений, следовательно, данный метод доказательства завершаемости не подходит для автоматического анализа.

Метод ограниченных систем

Метод основывается на доказанном утверждении о том, что любая ограниченная система завершаема [7]. Для доказательства символы исходной системы переписывания аннотируются натуральными числами — глубинами. Задаются они следующим образом: если минимальная глубина всех позиций строки на данном этапе — h, то каждая позиция в редукции будет иметь глубину $h+1$.

Далее необходимо доказать, что система является ограниченной: что для любого вывода в данной системе глубины будут глобально ограниченны. По сути это означает нахождение такого натурального числа c, что любая полученная глубина редукции будет меньше него.

Недостатком и, одновременно, достоинством, данного метода является работа только со строковыми системами переписывания. Вследствие этого при анализе кода программы его необходимо применять совместно с другими методами анализа на останов.

Проверка моделей

Для анализа на останов можно применить также методы проверки моделей. Недостатком их является сложность в составлении условий проверки и требование больших ресурсов памяти и процессора для выполнения самих проверок. Достоинством применения данных методов является возможность не просто доказать, завершается некоторый участок кода или нет, но и определить значения переменных, при которых может достигаться завершаемость или незавершаемость.

Для доказательства завершаемости может быть применён метод символьного выполнения: сначала необходимо проанализировать ГПУ программы с целью поиска циклов и условий их выполнения, затем построить граф символьного выполнения и проверить, что данные циклы могут выполняться бесконечно, т. е. условие выполнения цикла всегда верно. Надо понимать, что в худшем случае задача сведётся к полному перебору состояний переменных. Для уменьшения этого недостатка можно изначально исключить из анализа переменные, от которых условие выполнения цикла не зависит.

Завершаемость также может быть доказана с помощью доказательства LTL-формул. В некоторых работах предлагается критерий завершаемости для автомата Бюхи [8], который задаётся как ограничение пути на автомате. Недостатком такого подхода является как минимум сложность при транслировании исходной программы в автомат Бюхи. Также недостатком является изменение результатов при задании изменённой максимальной длины пути.

Рассмотренные в данной работе методы анализа на завершаемость не работают с кодом на языке программирования напрямую, а используют некоторую абстракцию. Наиболее быстрые и требующие меньшего количества ресурсов методы работают с математическими абстракциями, а следовательно, не учитывают ограничения реальных сред выполнения (как то разрядность, память вычислительной машины). В то же время методы из теории проверки моделей могут это учитывать, но в то же время требуют большого количества времени и ресурсов системы для анализа, что является минусом при использовании анализа на завершаемость в целях оптимизации кода. Таким образом, имеет смысл рассмотрение некоторой комбинации известных методов.

Список литературы

1. Koprowski A. Coq formalization of the higher-order recursive path ordering // Applicable Algebra in Engineering, Communication and Computing. 2009. Vol. 20. P. 379-425. DOI: 10.1007/s00200-009-0105-5.
2. Sarnat J., Schürmann C. Lexicographic Path Induction // 9th International Conference on Typed Lambda Calculi and Applications: proceedings. Berlin: Springer Berlin Heidelberg, 2009. Vol. 5608. P. 279 – 293.
3. Bagnara R., Mesnard F., Pescetti A., Zaffanella E. A New Look at the Automatic Synthesis of Linear Ranking Functions // Information and Computation. 2012. Vol. 215. P. 46-67.
4. Larraz D., Oliveras A., Rodríguez-Carbonell E., Rubio A. Proving Termination of Imperative Programs Using Max-SMT // Lecture Notes in Computer Science. 2014. Vol. 8559. P. 779-796. DOI: 10.1007/978-3-319-08867-9_52
5. Giesl J., Thiemann R., Schneider-Kamp P., Falke S. Mechanizing and Improving Dependency Pairs // Journal of Automated Reasoning. 2006. Vol. 37. P. 155–203. DOI: 10.1007/s10817-006-9057-7.
6. Zantema H. Termination of String Rewriting Proved Automatically // Journal of Automated Reasoning. 2005. Vol. 34. P. 105–139.
7. Geser A., Hofbauer D., Waldmann J. Termination Proofs for String Rewriting Systems via Inverse Match-Bounds // Journal of Automated Reasoning. 2005. Vol. 34. P. 365-385. DOI: 10.1007/s10817-005-9024-8.
8. Awedh M., Somenzi F. Termination Criteria for Bounded Model Checking: Extensions and Comparison // Electronic Notes in Theoretical Computer Science. 2006. Vol. 144. P. 51-66. DOI:10.1016/j.entcs.2005.07.019.