УДК 004.896
DOI: 10.15827/0236-235X.127.398-402

Дата подачи статьи: 01.02.19
2019. Т. 32. № 3. С. 398–402

Автоматизация верификации программ с использованием графоаналитических моделей вычислительного процесса

А.Г. Зыков [1], *к.т.н., тьютор, zykov_a_g@mail.ru*
Я.С. Голованев [1], *студент, golovanev98@mail.ru*
В.И. Поляков [1], *к.т.н., доцент, v_i_polyakov@mail.ru*

[1] *Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики (Университет ИТМО), г. Санкт-Петербург, 197101, Россия*

Постоянный рост объемов и количества создаваемого ПО требует новых инструментов, позволяющих сократить время на проектирование и разработку очередного продукта. В их число входят и средства автоматизации верификации. Верификация вычислительных процессов, реализованных программно, является сложной и трудоемкой задачей. Необходимость в новых инструментах автоматизации верификации возрастает из-за увеличения количества систем, использующих различные языки программирования, и требований к сокращению сроков реализации проектов. Актуальность задачи создания универсальных межязыковых средств верификации до сих пор высока.

В работе рассматриваются метод и средства автоматизации верификации вычислительных процессов на основе описания графоаналитической модели. Предлагаемый метод заключается в следующем. По разработанной программистом программе восстанавливается описание на разработанном языке и сравнивается с эталонным описанием графоаналитической модели, по которому эта модель создавалась; далее в автоматическом режиме по результатам сравнения либо программа верифицируется и определяется корректной, либо выдается детальная информация о наличии несовпадения; в интерактивном режиме исходный текст программы модифицируется с учетом полученной информации, процесс верификации повторяется.

Целью исследования является автоматизация верификации программ на языке C/C# по группе описаний графоаналитической модели вычислительного процесса.

В рамках данного исследования было создано средство, позволяющее преобразовывать исходные коды программ в описания графоаналитической модели и выполнять автоматизированную формальную верификацию проекта.

Разработанная утилита была проверена на восстановленных описаниях графоаналитической модели программ на C/C# и Java для обработки массивов (сортировка слиянием, алгоритм Дейкстры). Синтезированный исполняемый модуль был успешно протестирован в окружении операционной системы Windows 10.

В дальнейшем планируется развивать утилиту вместе с новыми версиями языка описания, чтобы расширить возможности анализа и верификации программ.

Ключевые слова: *графоаналитическая модель, синтаксический анализ, вычислительный процесс, конвертация программ, язык описания графоаналитической модели, автоматизация, верификация.*

Увеличение объемов проектирования и разработки программного продукта, повышенные требования к его надежности и достоверности требуют все больше средств и усилий для верификации программ на всех этапах жизненного цикла. Снижение трудоемкости данной процедуры невозможно без автоматизации методов анализа и принятия решения о соответствии разработанного программного продукта спецификации проекта. Работы в этом направлении ведутся во всем мире с использованием различных методов и средств [1–3]. В частности, в работе [4] рассматривается автоматизация верификации C-программ с использованием символического метода элиминации инвариантов цикла. Авторами предложен новый подход к верификации вычислительных процессов на основе *графоаналитических моделей* (ГАМ) вычислительного процесса. В работе [5] отмечается, что ГАМ является концентрированным описанием проектируемого программного продукта, формализацией и алгоритмизацией технического задания (спецификации проекта). Разработанный язык описания ГАМ [6, 7] и транслятор позволяют проводить анализ вычислительного процесса по машинному пред-

ставлению ГАМ вне зависимости от языковой платформы его реализации. Предложенный ранее метод формальной верификации вычислительного процесса на базе ГАМ [5] лег в основу разработанных алгоритмов и утилиты автоматизированной формальной верификации программ.

Автоматизация формальной верификации программ

Основные этапы верификации заключаются в восстановлении описания ГАМ по коду разработанной программы, сравнении его с эталонным описанием ГАМ, коррекции при необходимости исходного кода программы, после чего этапы повторяются до принятия решения об успешной верификации. Все возрастающий объем разрабатываемого программного продукта и повышение требований к ускорению цикла проектирования с обязательной верификацией проекта на каждом этапе невозможны без автоматизации принятия решения о верификации программного продукта.

Целью работы являются исследование и разработка методов и средств, позволяющих решить данную проблему на формальном уровне с минимальной зависимостью от языковых средств реализации проекта. Предложенный подход при исследовании доступной научной литературы не обнаружен, что свидетельствует об оригинальности разработанных метода и средства. Основные положения и реализация предлагаемого метода продемонстрированы на примере. Конвертация программы на языке C/C# с использованием синтаксического анализатора Roslyn [8, 9], имеющего открытый код, была описана в [7]. Утилита расширена возможностями восстановления описания ГАМ для программ на языке Java и блоком автоматизированного анализа и формальной верификации. Для проверки работоспособности созданного средства была разработана ГАМ сортировки слиянием. Выполнено описание ГАМ на входном языке, разработана программа на языке C, с помощью утилиты восстановлено языковое описание ГАМ, в интерактивном режиме выполнена автоматизированная верификация программы с соответствующей коррекцией специально внесенной ошибки.

Составленное описание ГАМ:

```
-- mergeSort(int array[]) :
LV1 (-100;2) { int n = array.length; }
CV2 (1;3:-1000) { n > 1 }
LV3 (2;4) { int middle = n / 2; int leftLength = middle; int rightLength = n - leftLength; int left[] = new int[leftLength]; int right[] = new int[rightLength]; int index = 0; }
UV4 (3,6;5)
CV5 (4;6:7) { index < middle }
LV6 (5;4) { left[index] = array[index]; index++; }
LV7 (5;8) { int rightIndex = 0; }
UV8 (7,10;9)
CV9 (8;10:11) { index < n }
LV10 (9;8) { right[rightIndex] = array[index]; rightIndex++; index++; }
LVF11 (9;12) { mergeSort(left); }
LVF12 (11;13) { mergeSort(right); }
LVF13 (12; -1000) { merge(array, left, right); }

-- merge(int array[], int right[], int left[]) :
LV1 (-100;2) { int leftIndex = 0; int rightIndex = 0; int resultIndex = 0; }
UV2 (1,6,9;3)
CV3 (2;4:-1000) { resultIndex < array.length }
CV4 (3;5:7) { leftIndex >= left.length }
UV5 (4,10;6)
LV6 (5;2) { array[resultIndex] = right[rightIndex]; rightIndex++; resultIndex++; }
CV7 (4;8:10) { rightIndex >= right.length }
UV8 (7,10;9)
LV9 (8;2) { array[resultIndex] = left[leftIndex]; leftIndex++; resultIndex++; }
CV10 (7;8:5) { left[leftIndex] < right[rightIndex] }
```

Написанная программа:

```
public static void mergeSort(int[] array) {
  int n = array.length;
  if (n > 1) {
    int middle = n / 2;
    int leftLength = middle;
    int rightLength = n - leftLength;
    int left[] = new int[leftLength];
    int right[] = new int[rightLength];
    int index = 0;
    while (index < middle) {
      left[index] = array[index];
      index++;
    }
    int rightIndex = 0;
    while (index < n) {
      right[rightIndex] = array[index];
      rightIndex++;
      index++;
    }
    mergeSort(left);
    mergeSort(right);
    merge(array, left, right);
  }
}
```

```
public static void merge(int[] array, int[] left, int[]
right) {
  int leftIndex = 0;
  int rightIndex = 0;
  int resultIndex = 0;
  while (resultIndex < array.length) {
  if (leftIndex >= left.length) {
    array[resultIndex] = right[rightIndex];
    rightIndex++;
    resultIndex++;
    } else {
      if (rightIndex >= right.length) {
      array[resultIndex] = left[leftIndex];
      leftIndex++;
      resultIndex++;
      } else {
       if (left[leftIndex] > right[rightIndex]) {
    array[resultIndex] = right[rightIndex];
      rightIndex++;
      resultIndex++;
      } else {
          array[resultIndex] = left[leftIndex];
          leftIndex++;
          resultIndex++;
          }
      }
    }
  }
}
```

Спроектированная утилита с использованием синтаксического анализатора сканирует исходный код программы и генерирует языковое описание ГАМ на разработанном языке [6, 7].

Скриншот работы утилиты по восстановлению описания приведен на рисунке (см. http://www.swsys.ru/uploaded/image/2019-3/2019-3-dop/6.jpg).

Восстановленное описание:

```
-- mergeSort(int array[]) :
LV1 (-100;2) { int n = array.length; }
CV2 (1;3:-1000) { n > 1 }
LV3 (2;4) { int middle = n / 2; int leftLength = mid-
dle; int rightLength = n - leftLength; int left[] = new
int[leftLength]; int right[] = new int[rightLength]; int
index = 0; }
UV4 (3,6;5)
CV5 (4;6:7) { index < middle }
LV6 (5;4) { left[index] = array[index]; index++; }
LV7 (5;8) { int rightIndex = 0; }
UV8 (7,10;9)
CV9 (8;10:11) { index < n }
LV10 (9;8) { right[rightIndex] = array[index]; right-
Index++; index++; }
LVF11 (9;12) { mergeSort(left); }
LVF12 (11;13) { mergeSort(right); }
LVF13 (12; -1000) { merge(array, left, right); }

-- merge(int array[], int right[], int left[]) :
LV1 (-100;2) { int leftIndex = 0; int rightIndex = 0;
int resultIndex = 0; }
UV2 (1,6,9;3)
CV3 (2;4:-1000) { resultIndex < array.length }
CV4 (3;5:7) { leftIndex >= left.length }
UV5 (4,10;6)
LV6 (5;2) { array[resultIndex] = right[rightIndex];
rightIndex++; resultIndex++; }
CV7 (4;8:10) { rightIndex >= right.length }
UV8 (7,10;9)
LV9 (8;2) { array[resultIndex] = left[leftIndex]; left-
Index++; resultIndex++; }
CV10 (7;5:8) { left[leftIndex] > right[rightIndex] }
```

Утилита сравнения двух описаний анализирует и проверяет их эквивалентность, выводя на экран и в файл информацию о результатах совпадения или несовпадения с указанием номеров неэквивалентных вершин и их содержимого.

Полученное и эталонное описания первого метода совпадают, для второго информация о несовпадении представлена на скриншоте (см. http://www.swsys.ru/uploaded/image/2019-3/2019-3-dop/7.jpg).

После корректировки программы и восстановления описания повторное сравнение приводит к положительному результату, что позволяет сделать вывод о формальной верификации проекта.

Результат сравнения второго анализируемого метода представлен на скриншоте (см. http://www.swsys.ru/uploaded/image/2019-3/2019-3-dop/8.jpg).

Результаты исследования

В ходе исследования была разработана утилита автоматизированной верификации программ на языках C/C# и Java на основании группы описаний ГАМ. В основе утилиты лежат восстановление описания ГАМ по коду разработанной программы и автоматизированная формальная верификация с возможностью корректировки программы в интерактивном режиме. Созданная утилита позволяет автоматически выявлять несовпадение восстановленного по коду программы описания на предложенном языке с эталонным описанием ГАМ вычислительного процесса, «мертвый код», закладки и недекларированные возможности. Разработанная утилита вошла в состав проектируемой учебно-исследовательской САПР верификации вычислительных процессов.

Необходимо помнить, что без надежного оборудования формально и функционально верифицированные программы не всегда могут работать достоверно [10].

Работа выполнена при поддержке РФФИ, проект № 17-07-00700.

Литература

1. Heitmeyer C., Archer M., Bharadwaj R., Jeffords R. Tools for constructing requirements specifications: The SCR toolset at the age of ten. Comput Syst Sci Eng, 2005, vol. 20, pp. 19–35.

2. Ball T., Bounimova E., Cook B., Levin V., Lichtenberg J., McGarvey C., Ondrusek B., Rajamani S.K., Ustuner A. Thorough static analysis of device drivers. Proc. EuroSys Conf., ACM SIGOPS, Belgium, 2006, vol. 40, pp. 73–85. DOI: 10.1145/1217935.1217943.

3. Berard B., Bidoit M., Finkel A., Laroussinie F., Petit A., Petrucci L., Schnoebelen P. Systems and software verification: model-checking techniques and tools. Springer Science & Business Media, 2013. DOI: 10.1007/978-3-662-04558-9.

4. Кондратьев Д.А., Марьясов И.В., Непомнящий В.А. Автоматизация верификации С-программ с использованием символического метода элиминации инвариантов циклов // Моделирование и анализ информационных систем. 2018. Т. 25. № 5. С. 491–505. DOI: 10.18255/1818-1015-2018-5-491-505.

5. Зыков А.Г., Кочетков И.В., Поляков В.И., Чистиков Е.Г. Синтезирование программ на основе описания графоаналитической модели // Программные продукты и системы. 2017. Т 30. № 4. С. 561–566. DOI: 10.15827/0236-235X.120.561-566.

6. Зыков А.Г., Кочетков И.В., Поляков В.И., Чистиков Е.Г. Методы анализа вычислительного процесса по графоаналитической модели // Конгр. IS&IT: сб. тр. 2017. Т. 2. С. 121–129.

7. Зыков А.Г., Кочетков И.В., Чистиков Е.Г., Швед В.Г. Автоматизация генерации описания графоаналитической модели программы // Защита информации. Инсайд. 2017. № 4. С. 60–65.

8. The .NET Compiler Platform ("Roslyn"). URL: https://github.com/dotnet/roslyn (дата обращения: 29.01.18).

9. Turner A. C# and visual basic – use roslyn to write a live code analyzer for your api. URL: https://msdn.microsoft.com/en-us/magazine/dn879356.aspx (дата обращения: 29.01.19).

10. Kolomoitcev V.S., Bogatyrev V.A., Polyakov V.I. Efficiency of intellectual system of secure access in a phased application of means of protection considering the intersection of the sets of threat detection. Proc. Open Semantic Technologies for Intelligent Systems. Minsk, BSUIR Publ., 2019, pp. 315–320.

Software & Systems
DOI: 10.15827/0236-235X.127.398-402

Received 01.02.19
2019, vol. 32, no. 3, pp. 398–402

Automation of program verification using graph analytical models of a computational process

A.G. Zykov [1], *Ph.D. (Engineering), tutor, zykov_a_g@mail.ru*
Ya.S. Golovanev [1], *Student, golovanev98@mail.ru*
V.I. Polyakov [1], *Ph.D. (Engineering), Associate Professor, v_i_polyakov@mail.ru*

[1] *St. Petersburg National Research University Information Technology, Mechanics and Optics (ITMO University), St. Petersburg, 197101, Russian Federation*

Abstract. The constant growth in the volume and quantity of software being created requires development of new tools that reduce the time for designing and developing the next product. These tools include automated verification tools. Verification of computational processes implemented by software is a complex and time-consuming task. The need for new verification automation tools is increasing due to an increase in the number of systems using various programming languages and requirements for shortening the implementation of projects. So far the urgency of the task of creating universal interlanguage verification tools remains high.

The paper discusses the method and means of computational process verification automation based on the description of a graph analytical model. The proposed method assumes that a description in the developed language is restored according to the developed program and then it is compared with the reference description

of a graph analytical model, according to which it was created. After that, in the automatic mode, the program is either verified and determined as correct by comparison results, or detailed information about a mismatch is given and a program source text is modified interactively according to the information received, and the verification process is repeated.

The aim of the study is to automate verification of C/C# programs by a group of descriptions of a graph analytical model of a computational process.

This study includes a developing a tool that allows converting program source codes into descriptions of a graph analytical model and performing automated formal verification of a project.

The developed utility was tested on the recovered descriptions of the graph analytical model of C/C# and Java programs for array processing (merge sort, Dijkstra's algorithm). The synthesized executable was successfully tested in the Windows 10 operating system.

In the future, it is planned to develop the utility along with new versions of a description language in order to expand the analyzing options and program verifying.

Keywords: graph analytical model, syntactic analysis, computational process, program conversion, language for description of the graph analytical model, automation, verification.

Acknowledgements. *This work is financially supported by RFBR, project no. 17-07-00700.*

References

1. Heitmeyer C., Archer M., Bharadwaj R., Jeffords R. Tools for constructing requirements specifications: The SCR toolset at the age of ten. *J. of Computer Systems Science and Engineering*. 2005, no. 20, pp. 19–35.

2. Ball T., Bounimova E., Cook B., Levin V., Lichtenberg J., McGarvey C., Ondrusek B., Rajamani S.K., Ustuner A. Thorough static analysis of device drivers. *Proc. of EuroSys 2006. ACM SIGOPS Operating Systems Review*, no. 40, pp. 73–85, 2006. DOI: 10.1145/1217935.1217943.

3. Bérard B., Bidoit M., Finkel A., Laroussinie F., Petit A., Petrucci L., Schnoebelen P. *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer Science & Business Media, 2013. DOI: 10.1007/978-3-662-04558-9.

4. Kondratyev D.A., Maryasov I.V., Nepomnyashchy V.A. Automation of verification of C-programs using the symbolic method of eliminating cycle invariants. *Modeling and Analysis of Information Systems*. 2018, vol. 25, no. 5, pp. 491–505. DOI: 10.18255/1818-1015-2018-5-491-505 (in Russ.).

5. Zykov A.G., Kochetkov I.V., Polyakov V.I., Chistikov E.G. Synthesizing programs based on the description of the graph-analytical model. *Software & Systems*. 2017, vol. 30, no. 4, pp. 561–566. DOI: 10.15827/0236-235X.120.561-566 (in Russ.).

6. Zykov A.G., Kochetkov I.V., Polyakov V.I., Chistikov E.G. Methods of analysis of the computational process on the graph analytical model. Proc. *Congress on Intelligent Systems and Information Technology "IS & IT" 17*. 2017, vol. 2, pp. 121–129 (in Russ.).

7. Zykov A.G., Kochetkov I.V., Chistikov E.G., Shved V.G. Automation of the generation of the description of the graph-analytical model of the program. Information Security. *Inside*. 2017, no. 4, pp. 60–65 (in Russ.).

8. *The .NET Compiler Platform ("Roslyn").* Available at: https://github.com/dotnet/roslyn (accessed January 29, 2018).

9. Turner A. *C# and Visual Basic for Your Api.* Available at: https://msdn.microsoft.com/en-us/magazine/dn879356.aspx (accessed January 29, 2019).

10. Kolomoitcev V.S., Bogatyrev V.A., Polyakov V.I. Efficiency of intellectual system of secure access in a phased application of means of protection considering the intersection of the sets of threat detection. *Proc. Open Semantic Technologies for Intelligent Systems*. Minsk, BSUIR Publ., 2019, pp. 315–320.

Для цитирования

Зыков А.Г., Голованев Я.С., Поляков В.И. Автоматизация верификации программ с использованием графоаналитических моделей вычислительного процесса // Программные продукты и системы. 2019. Т. 32. № 3. С. 398–402. DOI: 10.15827/0236-235X.127.398-402.

For citation

Zykov A.G., Golovanev Ya.S., Polyakov V.I. Automation of program verification using graph analytical models of a computational process. *Software & Systems*. 2019, vol. 32, no. 3, pp. 398–402 (in Russ.). DOI: 10.15827/0236-235X.127.398-402.