УДК 004.896
DOI: 10.15827/0236-235X.120.561-566
Дата подачи статьи: 14.09.17
2017. Т. 30. № 4. С. 561–566

СИНТЕЗИРОВАНИЕ ПРОГРАММ НА ОСНОВЕ ОПИСАНИЯ ГРАФОАНАЛИТИЧЕСКОЙ МОДЕЛИ

А.Г. Зыков, к.т.н., zykov_a_g@mail.ru;
И.В. Кочетков, аспирант, melmacson@gmail.com;
В.И. Поляков, к.т.н., доцент, v_i_polyakov@mail.ru;
Е.Г. Чистиков, магистрант, frazer@list.ru
(Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики (Университет ИТМО), Кронверкский просп., 49, г. Санкт-Петербург, 197101, Россия)

Количество и объемы создаваемого ПО растут ежегодно. Это побуждает разработчиков к созданию новых инструментов, позволяющих сократить время на разработку очередного продукта, в том числе средств автоматизации тестирования. Необходимость в новых инструментах автоматизации тестирования растет по причине того, что увеличивается количество систем, использующих различные языки программирования. Актуальность задачи поиска универсальных межязыковых средств тестирования остается высокой до сих пор.

В работе рассматривается верификация вычислительных процессов на основе графоаналитической модели. Основная идея этого подхода заключается в том, что разработанная программа конвертируется в описание графоаналитической модели и сравнивается с эталонным описанием, по которому она и создавалась; далее по результатам сравнения программа либо признается корректной, либо отправляется на доработку.

Узким местом такого подхода являются сама разработка программы на основе графоаналитической модели и потенциальная итеративность процесса. Эту проблему предлагается решить с помощью создания утилиты синтеза программ по эталонным описаниям графоаналитической модели. В данной работе рассматривается алгоритм преобразования объектной модели описания графоаналитической модели в текстовое представление операторов и выражений языка C#.

Целью исследования является автоматизация синтеза программ на языке C# по группе описаний графоаналитической модели вычислительного процесса. В рамках данного исследования было создано средство, позволяющее преобразовывать описания графоаналитической модели в исходные коды программ.

Разработанная утилита была проверена на описаниях графоаналитической модели программы для обработки массивов (сортировка, поворот). Синтезированный исполняемый модуль успешно протестирован в окружении операционной системы Windows 10.

В дальнейшем планируется развивать утилиту вместе с новыми версиями языка описания для расширения возможностей синтезируемых программ.

Ключевые слова: *графоаналитическая модель, Roslyn, генерация программ, синтезирование программ, автоматизация, тестирование, верификация.*

Верификация *вычислительных процессов* (ВП) включает в себя анализ всех путей управляющего графа программы [1, 2]. Одной из моделей, на основе которой возможно производить верификацию ВП, является *графоаналитическая модель* (ГАМ).

Данная модель представляет собой концентрированное описание проектируемого программного продукта и служит основой для разработки программы и анализа ВП. Для ГАМ разработан формальный язык описания (версия 1) [3], который содержит библиотеку примитивных вершин и правила их соединения для описания алгоритмов любой сложности. Более сложные конструкции языка (например, циклы или switch-case-конструкции) представляются в виде комбинации простейших.

Анализ программы на основе ГАМ осуществляется в несколько этапов.

1. Описание ВП в формате ГАМ и составление его формального описания на *языке описания ГАМ* (ЯОГ).
2. Проектирование программы на языке C#.
3. Конвертация исходного кода программы в ЯОГ.
4. Сравнение формального (эталонного) и сконвертированного (реального) описаний; в случае расхождения в структурах описаний программа отправляется на доработку (повторяются этапы 2–4); в случае совпадения можно утверждать, что разработанная программа соответствует ГАМ и формальная верификация прошла успешно, на основе чего можно осуществлять приемку исполняемого модуля.

Схема анализа программы с использованием ЯОГ представлена на рисунке 1.

Очевидно, что при таком подходе большая часть времени уходит на процесс разработки программы, которая удовлетворяла бы эталонному описанию ГАМ. Это происходит по нескольким причинам. Во-первых, программа разрабатывается человеком. Во-вторых, процесс разработки может включать более одной итерации. Поэтому возникает потребность в автоматизации процесса создания исполнимых модулей. Это возможно ввиду наличия эталонного описания ГАМ, которое является исходным для синтезирования программы.

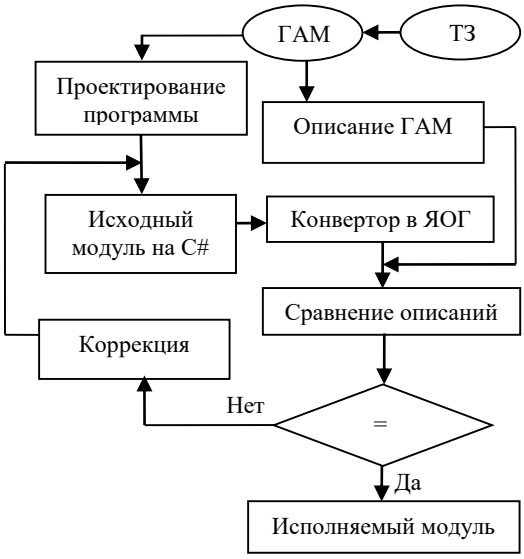


Рис. 1. Анализ программы с использованием ГАМ

Fig. 1. The analysis of the program using GAM

Разработка подобной утилиты позволит:

– экономить время на создание исполнимых модулей;

– создавать на базе ГАМ системы, которые дают возможность специалистам без знания языков программирования разрабатывать программы и тестировать их в реальной среде;

– избегать появления в программе недекларированных возможностей и «мертвого кода».

Предполагается получать исходный код программы на языке C# из группы описаний ГАМ в два этапа: преобразование объектных моделей описаний в текстовое представление выражений и конструкций языка C# [4] и дополнительная модификация (запаковка текстового представления в сущность класса, добавление используемых библиотек и другое) полученного исходного кода средствами синтаксического анализатора Roslyn [5].

Модифицированная схема синтеза программы на основе ГАМ ВП с использованием разработанной утилиты представлена на рисунке 2.

Целью данного исследования является автоматизация создания программ на языке C# на основе описания ГАМ. Разработанная утилита была проверена на описаниях программы обработки массивов (сортировка, поворот). Проверка включала в себя конвертацию описаний ГАМ в исходный код программы, компиляцию этого исходного кода средствами Visual Studio и MSBuild и тестирование программы в окружении операционной системы Windows 10.

В дальнейшем для расширения возможностей синтезируемых программ планируется развивать утилиту вместе с новыми версиями языка описания. Наряду с этим планируется развивать экосистему ГАМ в рамках учебно-исследовательской

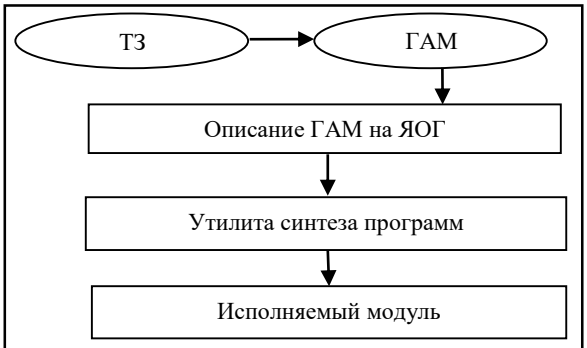


Рис. 2. Схема синтеза программы с использованием описания ГАМ ВП

Fig. 2. The scheme of the program synthesis using the description of GAM of a computing process

САПР верификации ВП, разрабатываемой на кафедре информатики и прикладной математики Университета ИТМО.

Формализация описания ГАМ

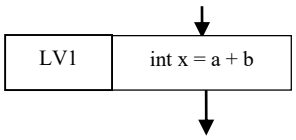
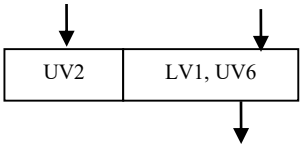
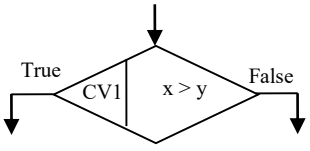
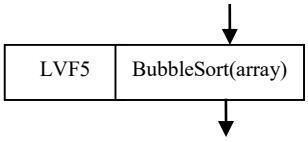
ЯОГ предназначен для универсального представления алгоритма программы в виде последовательности вершин трех типов: условной, линейной и объединяющей. Описание также содержит набор правил именования вершин и передачи управления. В таблице 1 приведены примеры изображения и описания вершин ГАМ.

Таблица 1

Примеры изображения и описания вершин ГАМ

Table 1

Examples of GAM tops representation and description

Вершина	Пример изображения	Описание
Линейная		LV1(-100; 2) {int x = a + b;}
Объединяющая		UV2(1,6; 3)
Условная		CV1(-100; 2:3) {x > y}
Вызов функции		LVF5 (4; 6) {Bubble-Sort(array)}

Базовые блоки (циклы, множественный выбор и др.) представляются в виде комбинации простей-

ших вершин. В таблице 2 приведен пример изображения и описания цикла со счетчиком, решающего задачу вычисления факториала числа *N*.

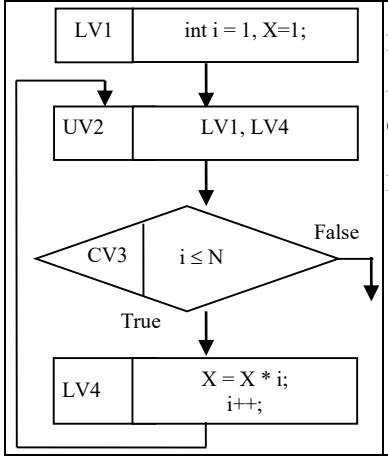
Необходимо отметить, что вход в ВП и выход из него несколько отличаются от обычной линейной вершины, что начальной и конечной вершинами являются такие вершины, номера которых – отрицательные числа. Вершина вызова функции используется в том случае, когда для этой функции существует описание ГАМ. Для функций из стандартных библиотек используется обычная линейная функция. При описании ГАМ в круглых скобках указываются номера входных и выходных вершин.

Таблица 2

Пример изображения и описания цикла со счетчиком

Table 2

The example of counting loop representation and description

Пример изображения	Описание
	LV1(-100;2) {int i = 1, X = 1;} UV2(1,4;3) CV3(2;4:5) {i ≤ N} LV4(3;2) {X = X * i; i++;}

ЯОГ может быть расширен: например, добавлен частный случай условной вершины – ждущая вершина [6].

Алгоритм преобразования

Синтезирование программ на основе описаний ГАМ будем производить по следующему алгоритму.

1. Последовательное чтение каждого описания ГАМ из указанной директории.

2. Преобразование описания ГАМ в отдельный метод программы [7]:

– получение объектной модели описания ГАМ;

– последовательное преобразование объектной модели в конструкции целевого языка программирования;

– дополнение обязательными атрибутами (имя, модификаторы доступа, возвращаемый результат, аргументы и др.).

3. Объединение всех полученных методов в единый файл.

4. Дополнение обязательными атрибутами полученного файла (инструкции включения, наименование класса и др.).

5. Компиляция полученного файла в исполняемый файл или библиотеку [8].

Прохождение путей исполнения начинается с начальной вершины. Прохождение конкретного пути прерывается в случаях достижения объединяющей вершины, достижения конца, обнаружения рекурсивного фрагмента.

При отсутствии вершины с требуемым идентификатором процесс преобразования прекращается.

Ввиду того, что конструкции в теле линейной вершины и условиях в текущей версии описания представляют свободный текст без семантики, синтез программы можно производить только на основании текстового представления, а не объектов [9]. Базовые управляющие вершины преобразовываются следующим образом:

– к линейной вершине добавляется тело линейного блока к представлению родительского блока без дополнительных преобразований;

– условная и объединяющая вершины не преобразовываются для языка C# и используются только в составе базовых блоков.

Основную сложность при преобразовании представляют базовые блоки, являющиеся совокупностью управляющих и линейных вершин, но представляющие единое целое. Как было отмечено ранее, существует всего несколько базовых блоков. Вершину, управление которой передается из базового блока, можно проследить по условной вершине, отвечающей за условие цикла (либо в конце блока, либо после первой вершины). После определения списка вершин и пути передачи управления можно произвести свертку базового блока в стандартную линейную вершину и вставить в ее тело текстовое представление блока [10]. На рисунке 3 представлена схема свертки при преобразовании.

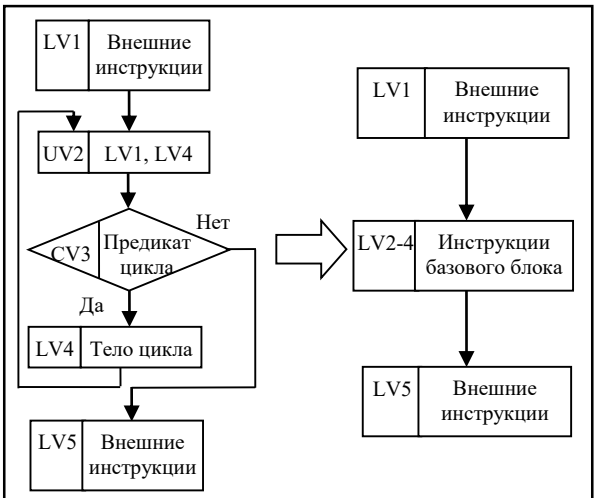


Рис. 3. Свертка базового блока

Fig. 3. A base unit fold

Результаты тестирования

Для проверки утилиты синтеза программ воспользуемся описаниями ГАМ для программы обработки массивов. Этот вычислительный процесс содержит два описания.

Описание ГАМ методом Main: static void Main(string[ ] args).

```
LV1(-100; 2) { var option = int.Parse(args[0]); var array = args[1].Split(new[] { "." }, StringSplitOptions.RemoveEmptyEntries).Select(int.Parse).ToArray(); }
CV2 (1; 3:4) { option == 0 }
LV3 (2; 7) { array = array.Reverse().ToArray() }
CV4 (2; 5:6) { option == 1 }
LVF5 (4; 6) { BubbleSort(array) }
UV6 (4,5; 7)
UV7 (3,6; 8)
LV8 (7; 9) { var enumerator = array.GetEnumerator() }
UV9 (8,11;10)
CV10 (9; 11:12) { enumerator.MoveNext() }
LV11 (10; 9) { var element = enumerator.Current; Console.Write(element + " "); }
LV12 (10; -10) { Console.ReadKey() }
```

Описание ГАМ методом BubbleSort: private static void BubbleSort(int[ ] arr).

```
LV1(-100; 2) { int temp = 0; int write = 0; }
UV2(1,11; 3)
CV3 (2; 4:-10) { write < arr.Length }
LV4 (3; 5) { int sort = 0 }
UV5 (4,10; 6)
CV6 (5; 7:11) { sort < arr.Length - 1 }
CV7 (6; 8:9) { arr[sort] > arr[sort + 1] }
LV8 (7; 9) { temp = arr[sort + 1]; arr[sort + 1] = arr[sort]; arr[sort] = temp; }
UV9 (7,8; 10)
LV10 (9; 5) { sort++ }
LV11 (6; 2) { write++ }
```

Синтезированная программа компилируется и выполняется корректно:

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Linq;
namespace SynthesizedProgram
{
    class Program
    {
        static void Main(string[] args)
        {
            var option = int.Parse(args[0]);
            var array = args[1].Split(new[] { "." }, StringSplitOptions.RemoveEmptyEntries).Select(int.Parse).ToArray();
            if (option = = 0)
            {
                array = array.Reverse().ToArray();
            }
            else
            {
                if (option == 1)
                {
                    BubbleSort(array);
                }
            }
            var enumerator = array.GetEnumerator();
            while (enumerator.MoveNext())
            {
                var element = enumerator.Current;
                Console.Write(element + " ");
            }
            Console.ReadKey();
        }

        private static void BubbleSort(int[] arr)
        {
            int temp = 0;
            int write = 0;
            while (write < arr.Length)
            {
                int sort = 0;
                while (sort < arr.Length – 1)
                {
                    if (arr[sort] > arr[sort + 1])
                    {
                        temp = arr[sort + 1];
                        arr[sort + 1] = arr[sort];
                        arr[sort] = temp;
                    }
                    sort++;
                }
                write++;
            }
        }
    }
}
```

Исполнимый модуль был протестирован с помощью средств, описанных в [11].

Недостатки описания ГАМ

В ходе исследования было выявлено несколько проблем описания ГАМ, затрудняющих синтез более сложных программ.

1. Свободная форма содержания линейных блоков и условий. По этой причине невозможно производить преобразование на основе синтаксических деревьев двух языков (описания ГАМ и целевого). Требуется наличие стандартных понятий любого языка программирования (система типов, операторы, выражения и др.).

2. Отсутствие данных о зависимостях получаемой программы (выражения Using). На текущем этапе их можно заполнить определенным набором, покрывающим требуемый спектр возможностей, или пытаться автоматически определить по исходному коду, как это делается в современных интегрированных средах разработки (Integrated Development Environment).

3. Отсутствие сигнатуры описываемого метода. На текущем этапе ее можно определить только по названию файла, содержащего описание, что является неустойчивым решением и может приводить к сбоям.

Ограничения утилиты синтезирования программы

Из перечисленных недостатков и особенностей языка C# вытекают ограничения, накладываемые на исходные описания:

– абсолютная корректность содержимого блоков из-за невозможности его контролировать;

– невозможность использования дополнительных сущностей (классов, перечислений и пр.), так как их определения не входят в стандартное описание ГАМ;

– необходимость специального описания библиотек, не входящих в стандартный набор;

– совпадение названия файла описания с полной сигнатурой описываемого метода;

– наличие описания метода Main для исполняемых программ;

– невозможность использования условных и объединяющих вершин вне состава базовых блоков.

Результаты исследования

В ходе исследования был разработан прототип утилиты синтезирования программ на языке C# на основании группы описаний ГАМ. В основе утилиты лежит преобразование объектной модели описания ГАМ в текстовое представление выражений и операторов языка C#. Такой подход является следствием того, что текущая версия описания ГАМ поддерживает только свободную форму записи содержимого управляющих вершин и нет возможности производить преобразование на основе синтаксических деревьев [12]. Проверка корректности и дополнительная модификация (using, namespaces и др.) синтезируемых исходных кодов производятся с помощью синтаксического анализатора Roslyn.

На выходе утилиты получается файл с исходным кодом cs-формата. Он состоит из единственного класса, содержащего методы, полученные путем преобразования входящих описаний. Этот файл опционально может быть скомпилирован либо в динамическую библиотеку (.dll), либо в исполняемый модуль (.exe).

Разработанная утилита имеет командный интерфейс, с помощью которого предполагается в будущем использовать ее в экосистеме ГАМ в качестве одного из синтезаторов программ. Вместе с развитием языка описания будет произведен переход от текстового преобразования на преобразование на основе синтаксических деревьев с применением Roslyn. Также предполагается использовать возможности Roslyn для постобработки исходных кодов, их оптимизации, статической верификации и для других задач [13, 14].

Работа выполнена при финансовой поддержке РФФИ, проект № 17-7-00700.

Литература

1. Немолочнов О.Ф., Зыков А.Г., Поляков В.И. [и др.]. Верификация в исследовательских, учебных и промышленных системах // Науч.-технич. вестн. СПбГУ ИТМО. Вып. 11. Актуальные проблемы анализа и синтеза сложных технических систем. 2003. С. 146–151.

2. Bérard B., et al. Systems and software verification: model-checking techniques and tools. Springer Science & Business Media, 2013.

3. Зыков А.Г., Поляков В.И., Чистиков Е.Г., Кочетков И.В. Формализация анализа программной реализации вычислительного процесса в САПР // IS&IT'16: тр. Конгресса по интеллект. системам и информ. технологиям. Таганрог: Изд-во ЮФУ, 2016. Т. 1. С. 69–75.

4. Parr T. The definitive ANTLR 4 reference. Pragmatic Bookshelf, 2 ed., 2013, 326 p.

5. The .NET Compiler Platform ("Roslyn"). URL: https://github.com/dotnet/roslyn (дата обращения: 10.05.17).

6. Зыков А.Г., Немолочнов О.Ф., Поляков В.И., Безруков А.В., Кузьмин В.В. Графо-аналитические модели как средство верификации вычислительных процессов // Междунар. конгресс по интеллект. системам и информ. технологиям: сб. тр. М.: Физматлит, 2010. Т. 2. С. 140–145.

7. Wang P., et al. Translingual obfuscation. Security and privacy (EuroS&P), 2016. IEEE Europ. Sympos. 2016, pp. 128–144.

8. Bock J. .NET Development Using the Compiler API. Apress, 2016.

9. Ilyushin E., Namiot D. On source-to-source compilers. Intern. Jour. of Open Information Technologies, 2016, vol. 4, no. 5.

10. Bates C., Robinson A. APRT–another pattern recognition tool. GSTF JoC, 2017, vol. 5, no. 2.

11. Зыков А.Г., Кочетков И.В., Поляков В.И. Применение системы KLEE для автоматизации тестирования программ на языках C/C++ // Программные продукты и системы. 2016. Т. 29. № 4. С. 101–106. DOI: 10. 15827/0236-235X.116.101-106.

12. Zhu Z., et al. BiYacc: Roll your parser and reflective printer into one. 4th Intern. Workshop on Bidirectional Transformations co-located with Software Technologies: Applications and Foundations (STAF 2015). CEUR Workshop Proc., 2015, vol. 1396, pp. 43–50.

13. 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: 29.01.17).

14. Зыков А.Г., Кочетков И.В., Чистиков Е.Г., Швед В.Г. Автоматизация генерации описания графо-аналитической модели программы // Защита информации. Инсайд. 2017. № 4. С. 2–7.

Software & Systems
DOI: 10.15827/0236-235X.120.561-566

Received 14.09.17
2017, vol. 30, no. 4, pp. 561–566

PROGRAM SYNTHESIZING BASED ON A GRAPH-ANALYTIC MODEL DESCRIPTION

A.G. Zykov [1], *Ph.D. (Engineering), zykov_a_g@mail.ru*
I.V. Kochetkov [1], *Postgraduate Student, melmacson@gmail.com*
V.I. Polyakov [1], *Ph.D. (Engineering), Associate Professor, v_i_polyakov@mail.ru*
E.G. Chistikov [1], *Graduate Student, frazer@list.ru*

[1] *The National Research University of Information Technologies, Mechanics and Optics, Kronverksky Ave. 49, St. Petersburg, 197101, Russian Federation*

Abstract. The quantity and volumes of the developed software grow annually. It stimulates developers to create new tools enabling to reduce time for the next product development. It also includes testing automation equipment. The demand for new instruments of test automation increases due to increasing number of systems using different programming languages. The relevance of the task of searching universal cross-language testing tools remains high.

The paper considers verification of computing processes based on a graph-analytic model (GAM). The key idea of this approach is that the developed program is converted into a GAM description and is compared to the reference GAM description according to which it was created. Further, according to the results of comparing, the program either is recognized as correct, or is sent back for revision.

A bottle neck of such approach is development of the program based on GAM and a potential iteration nature of the process. The authors suggest a special utility to solve this problem. This utility performs synthesis of programs for reference descriptions. The paper considers an algorithm of conversion of a GAM description object model into text representation of C# operators and expressions.

A research objective is automation of program synthesis in C# by a group of GAM descriptions of a computing process. Within the research, we have created a tool enabling to transform GAM descriptions into program source codes.

We have checked the developed utility on GAM descriptions of an array processing program (sorting, turn). The synthesized executed module has been successfully tested in Windows 10 operating system environment.

In the future we plan to develop the utility along with new versions of a description language to enrich the possibilities of synthesizable programs.

Keywords: graph-analytic model, Roslyn, generation of programs, program synthesis, automation, testing, verification.

Acknowledgements. *This work is financially supported by RFBR, project no. 17-7-00700.*

References

1. Nemolochnov O.F., Zykov A.G., Polyakov V.I., Lazdin A. Verification in research, training and industrial systems. *Nauch.-tekhnich. vestnik SPbGU ITMO* [Scientific and Technical Journal of Information Technologies, Mechanics and Optics]. St. Petersburg, SPbGU ITMO Publ., 2003, no. 5 (11), pp. 146–151 (in Russ.).

2. 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 Publ., 2013.

3. Zykov A.G., Polyakov V.I., Chistikov E.G., Kochetkov I.V. Formalization of the analysis of computation process software implementation in CAD. *Tr. Kongressa po intellektualnym sistemam i informatsionnym tekhnologiyam "IS&IT'16"* [Proc. Congr. on Intelligent Systems and Information Technologies "IS&IT"]. Taganrog, YuFU Publ., 2016, vol. 1, pp. 69–75 (in Russ.).

4. Parr T. *The definitive ANTLR 4 Reference*. Pragmatic Bookshelf Publ., 2013, 326 p.

5. *The .NET Compiler Platform ("Roslyn")*. Available at: https://github.com/dotnet/roslyn (accessed May 10, 2017).

6. Zykov A.G., Nemolochnov O.F., Polyakov V.I., Bezrukov A.V., Kuzmin V.V. Graph-analytic models as a verification facility of computing processes. *Tr. Mezhdunar. kongr. po intellektualnym sistemam i informatsionnym tekhnologiyam* [Proc. Int. Congr. on Intelligent Systems and Information Technologies]. Moscow, Fizmatlit Publ., 2010, vol. 2, pp. 140–145 (in Russ.).

7. Wang P., Wang S., Ming J., Jiang Y., Wu D. Translingual obfuscation. *2016 IEEE European Symp. on Security and Privacy (EuroS&P)*. 2016, pp. 128–144.

8. Bock J. *.NET Development Using the Compiler API*. Apress, 2016.

9. Ilyushin E., Namiot D. On source-to-source compilers. *Int. Jour. of Open Information Technologies*. 2016, vol. 4, no. 5, pp. 48–51.

10. Bates C., Robinson A. APRT–Another Pattern Recognition Tool. *GSTF Jour. on Computing (JoC)*. 2017, vol. 5, no. 2, pp. 46–52.

11. Zykov A.G., Kochetkov I.V., Polyakov V.I. KLEE for automatic testing programs in C/C++. *Programmnye produkty i sistemy* [Software & Systems]. 2016, no. 4 (29), pp. 101–106 (in Russ.).

12. Zhu Z. BiYacc: Roll your parser and reflective printer into one. *CEUR Workshop Proc. 4th Int. Workshop on Bidirectional Transformations co-located with Software Technologies: Applications and Foundations (STAF 2015)*. 2015, vol. 1396, pp. 43–50.

13. Turner A. *C# and Visual Basic – Use Roslyn to Write a Live Code Analyzer for Your Api*. Available at: https://msdn.microsoft.com/en-us/magazine/dn879356.aspx (accessed January 29, 2017).

14. Zykov A.G., Kochetkov I.V., Chistikov E.G., Shved V.G. Automation of program graph-analitic model description generation. *Zaŝita informacii. Inside* [Information Security. Inside]. 2017, no. 4, pp. 2–7 (in Russ.).