СБОРНИК **НАУЧНЫХ ТРУДОВ НГТУ. – 2014. – № 3(77).** – 73–80

СОВРЕМЕННЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

УДК 62-50:519.216

СРАВНЕНИЕ МЕТОДОВ АБСТРАКТНОЙ ИНТЕРПРЕТАЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ*

А.А. ВОЕВОДА[1], Д.О. РОМАННИКОВ[2]

[1] *630073, РФ, г. Новосибирск, пр. Карла Маркса, 20, Новосибирский государственный технический университет, доктор технических наук, профессор. E-mail: ucit@ucit.ru*
[2] *630073, РФ, г. Новосибирск, пр. Карла Маркса, 20, Новосибирский государственный технический университет, кандидат технических наук, старший преподаватель кафедры автоматики. E-mail: rom2006@gmail.com*

В работе приводится сравнительный анализ двух методов абстрактной интерпретации: один из них был предложен P. Cousot, второй разработан на кафедре автоматики НГТУ независимо от первого. Оба метода основываются на общей идее преобразования исходного кода программы в модель и последующего ее анализа. Оба метода предполагают исследование на всех возможных путях исполнения программы. В работе приведены примеры анализа одной и той же программы исследуемыми методами, выделены различия в исследуемых методах. В частности, метод абстрактной интерпретации предполагается использовать для автоматической проверки одного класса ошибок, для которого используется специальный домен для записи данных. Второй исследуемый метод позволяет разработчику выполнять произвольные проверки, а те ошибки, которые определялись доменом, могут быть выявлены в момент интерпретации программы. Другим отличием является то, что метод абстрактной интерпретации позволяет анализировать исходный код программ даже при их неполном представлении, несмотря на то что это негативно сказывается на точности анализа. Для второго исследуемого метода необходимо иметь полное представление исходного кода.

Ключевые слова: программное обеспечение, тестирование, входные интервалы, формальная верификация, динамическая верификация, верификация, проверка моделей, модели программного обеспечения, графы, тотальная корректность программ

* Статья получена 8 августа 2014 г.

Работа выполнена при финансовой поддержке Минобрнауки России по государственному заданию № 2014/138. Тема проекта «Новые структуры, модели и алгоритмы для прорывных методов управления техническими системами на основе наукоемких результатов интеллектуальной деятельности».

ВВЕДЕНИЕ

Для поиска ошибок в программном обеспечении (ПО) традиционно в индустрии разработки ПО используется тестирование. Несмотря на то что тестирование обладает рядом недостатков: существенная трудоемкость, влияние человеческого фактора, невозможность обеспечить полноту покрытия сценариев и входных данных, оно является наиболее доступным средством проверки ПО, в силу того что на современном этапе развития формальные методы (статический анализ, проверка моделей, абстрактная интерпретация и др.) [1–4] не позволяют заменить ручного тестирования из-за ограничения вычислительных ресурсов и несовершенства алгоритмов анализа. Тем не менее методы анализа ПО [1–10] ограниченно применяются в разработке в сферах деятельности, где стоимость ошибки чрезвычайно высока. Также намечается тенденция, когда инструменты верификации, а также инструменты, основанные на формальных методах, используются для анализа прикладного ПО, например верификатор от Microsoft – для проверки драйверов сторонних устройств и др. Поэтому развитие формальных методов анализа ПО имеет большое значение.

В данной статье рассматривается сравнение двух методов абстрактной интерпретации[1] [1, 2]. Данные методы были получены независимо друг от друга и основаны на общей идее, но также содержат и некоторые различия, которые будут рассмотрены далее в работе.

ОБЗОР СРАВНИВАЕМЫХ МЕТОДОВ

Метод абстрактной интерпретации предложен П. Коусотом (P. Cousot) в работе [1] и активно развивался в следующие годы, в результате чего он лег в основу многих других методов и инструментов [3, 4]. Основной идеей предложенного метода является проверка ПО по следующей процедуре: 1) нахождение возможных путей, по которым может проходить программа; 2) представление программы в виде специальной семантики; 3) представление данных в виде домена, на котором необходимо выполнить проверку; 4) анализ ПО на основе составленных путей и представления данных.

Пример такого анализа приведен на рис. 1, где анализируется небольшая программа (в верхнем правом углу рисунка). Семантика модели программы

[1] В обеих оригинальных работах методы назывались абстрактной интерпретацией, но поскольку метод в работе [1] предложен значительно раньше, то его далее будем называть оригинальным названием, а метод в работе [2] – исследуемым методом или предлагаемым методом.

представлена в верхней части рисунка посередине, а в остальных частях представлены начальные и конечные стадии анализа. В начальном состоянии (на первой строке) переменная *x* (запись X_1 следует читать как значение переменной *x* в строке 1) не определено – 0. Далее при исполнении программы значение изменяется на интервал [1, 1] и управление переходит в следующую строку. После входа в цикл *x* будет постоянно инкрементироваться до момента последней итерации цикла, т. е. конечное значение *x* будет равно 10000, а X_3 принимает значения в интервале [2, 10000]. После выхода из цикла X_4 = [10000, 10000].

x := 1 while x < 10000 do x := x + 1 od;	X_1 = [1, 1]; X_2 = ($X_1 \cup X_3$) ∩ [-∞, 9999]; X_3 = $X_2 \oplus$[1,1]; X_4 = ($X_1 \cup X_3$) ∩ [10000, +∞].	X_1 = 0; X_2 = 0; X_3 = 0; X_4 = 0.
X_1 = [1, 1]; X_2 = 0; X_3 = 0; X_4 = 0.	X_1 = [1, 1]; X_2 = [1, 1]; X_3 = 0; X_4 = 0.	X_1 = [1, 1]; X_2 = [1, 1]; X_3 = [2, 2]; X_4 = 0.
X_1 = [1, 1]; X_2 = [1, 2]; X_3 = [2, 2]; X_4 = 0.	X_1 = [1, 1]; X_2 = [1, 2]; X_3 = [2, 3]; X_4 = 0.	X_1 = [1, 1]; X_2 = [1, 3]; X_3 = [2, 3]; X_4 = 0.
X_1 = [1, 1]; X_2 = [1, +∞]; X_3 = [2, +∞]; X_4 = 0.	X_1 = [1, 1]; X_2 = [1, 9999]; X_3 = [2, +10000]; X_4 = 0.	X_1 = [1, 1]; X_2 = [1, 9999]; X_3 = [2, +10000] X_4 = [+10000, +10000].

Рис. 1. Пример анализа программы методом абстрактной интерпретации

В работе [2] предложен метод, который также заключается в абстрактной интерпретации программы, но был разработан позднее. Данный способ предполагает следующую процедуру: 1) вычисление всех возможных путей выполнения программы; 2) запись всех возможных вариантов переменных, полученных в результате исполнения оператором; 3) проверка того, что проверяемое условие исполняется на всех возможных вариантах переменных, полученных на предыдущем шаге.

x := 1 while x < 10000 do x := x + 1 od; assert(x ≥ 10000)	S1: x := 1 S2: while x < 10000 do S3: x := x + 1 S4: od; assert(x ≥ 10000)
P = P1 = S1 ∩ S2 ∩ S3 ∩ S4.	S1 = {x} = {1}.
S1 = {1}; S2 = {1}.	S1 = {1}; S2 = {1}; S3 = {2}.
S1 = {1}; S2 = {1}; S3 = {3};	S1 = {1}; S2 = {1}; S3 = {10000}.
S1 = {1}; S2 = {1}; S3 = {10000}; S4 = {10000}.	for s in S4 assert(x(s) ≥ 10000) ---------- assert(10000 ≥ 10000)

Рис. 2. Пример анализа программы методом из [2]

Пример такого анализа приведен на рис. 2, где исследуется аналогичная программа, как и на рис. 1. В начале исполнения программы значение x в S1 = {1}. При входе в цикл значение переменной в S2 изменяется от {2} до {10000}. Значение переменной в S4 равно {10000}. Кроме того, в данном методе используются условия в качестве дополнительных ограничений и информации для анализа.

СРАВНЕНИЕ МЕТОДОВ

Оба метода [1, 2] основаны на общей идее преобразования программы в определенную семантику и последующего определения значений переменных. В методах абстрактной интерпретации и входных интервалов имеются некоторые отличия.

Назначение. Абстрактная интерпретация предназначена для проверки программы на конкретный вид ошибок, который зависит от домена. Например, для проверки выхода за пределы границы массива может применяться интервальный домен. Назначение предлагаемого метода заключается в проверке условий, заданных разработчиком в программе, т. е. в проверке произвольных условий, а не только заданных. С другой стороны, поскольку процесс

анализа по своей сути является интерпретацией программы, то во время анализа ошибки исполнения программы будут проявляться, что приведет к остановке анализа на заданном пути при выбранных начальных условиях.

Требования по полноте кода. В настоящее время программы часто взаимодействуют с другими программными устройствами, при этом они обмениваются данными. Метод абстрактной интерпретации более толерантен к отсутствию полной информации о коде программы, т. е. при анализе сетевых взаимодействий нет необходимости иметь полную модель, включая исходный код сторонних устройств. Несомненно, это сказывается на точности выполняемого анализа и приводит к наличию ложных срабатываний или пропуску ошибок. Предлагаемый метод предполагает полную информацию об исследуемой системе.

Семантика. Данное различие носит технический характер, который напрямую не влияет на анализ. Модель в абстрактной интерпретации имеет семантику, позволяющую выполнять анализ, не прибегая к исходному коду. Во втором исследуемом методе анализ выполняется на основе исходного кода, а семантика модели применяется для записи возможных вариантов исполнения программы.

ВЫВОД

Исходя из описания исследуемых методов, примеров их применения, а также их сравнения можно сделать вывод, что данные методы имеют существенные различия. В результате исследуемый метод значительно увеличивает возможности по проверке свойств ПО, но при этом не достигается автоматическая проверка на набор ошибок (например, выход за границы массива). Различия в семантике модели, на сегодняшний взгляд, не кажутся основательными, так как обе семантики непосредственного влияния на конечный анализ не имеют.

СПИСОК ЛИТЕРАТУРЫ

1. *Cousot P., Cousot R.* Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints // Conference record of the fourth ACM symposium on principles of programming languages, Los Angeles, California, Jan. 17–19, 1977. – New York: ACM Press, 1977. – P. 238–252.

2. *Воевода А.А., Романников Д.О., Зимаев И.В.* Применение UML диаграмм и сетей Петри при разработке встраиваемого программного обеспечения // Научный вестник НГТУ. – 2009. – № 4 (37). – С. 169–174.

3. *Кларк Э., Грамберг О., Пелед Д.* Верификация моделей программ: Model checking. – М.: МЦНМО, 2002. – 416 с.

4. *Bush W.R., Pincus J.D., Sielaff D.J.* A static analyzer for finding dynamic programming errors // Software: practice and experience. – 2000. – Vol. 30, iss. 7. – P. 775–802. – doi: 10.1002/(SICI)1097-024X(200006)30:7<775::AID-SPE309>3.0.CO;2-H.

5. *Глухих М.И., Ицыксон В.М., Цесько В.А.* Использование зависимостей для повышения точности статического анализа программ // Моделирование и анализ информационных систем. – 2011. – Т. 18, № 4. – С. 68–79.

6. *Романников Д.О.* О поиске входных интервалов // Сборник научных трудов НГТУ. – 2014. – № 1 (75). – С. 140–145.

7. Coherent clusters in source code / S. Islam, J. Krinke, D. Binkley, M. Harman // Journal of systems and software. – 2014. – Vol. 88. – P. 1–24.

8. *Марков А.В., Романников Д.О.* Алгоритм трансляции диаграммы активности в сеть Петри // Доклады Академии наук высшей школы Российской Федерации. – 2014. – № 1 (22). – С. 104–112.

9. *Романников Д.О.* Нахождение ошибок обращения к несуществующим элементам массива на основании результатов анализа сети Петри // Сборник научных трудов НГТУ. – 2012. – № 1 (67). – С. 115–120.

10. *Воевода А.А., Марков А.В., Романников Д.О.* Разработка программного обеспечения: проектирование с использованием UML диаграмм и сетей Петри на примере АСУ ТП водонапорной станции // Труды СПИИРАН. – 2014. – Вып. 3 (34). – С. 218–231.

Воевода Александр Александрович – доктор технических наук, профессор кафедры автоматики НГТУ. Основное направление научных исследований – управление многоканальными объектами. Имеет более 200 публикаций. E-mail: ucit@ucit.ru

Романников Дмитрий Олегович – кандидат технических наук, старший преподаватель кафедры автоматики НГТУ. Основное направление научных исследований – формальная верификация, проверка моделей. Имеет 31 публикацию. E-mail: rom2006@gmail.com

Comparison of abstract interpretation methods of software*

A.A. Voevoda[1], D.O. Romannikov[2]

[1] *Novosibirsk State Technical University, 20 Karl Marks Avenue, Novosibirsk, 630073, Russian Federation, doctor of Technical Sciences, professor. E-mail: ucit@ucit.ru*
[2] *Novosibirsk State Technical University, 20 Karl Marks Avenue, Novosibirsk, 630073, Russian Federation, candidate of Technical Sciences, senior lecturer at the department of automation. E-mail: rom2006@gmail.com*

Comparative analysis of two methods of abstract interpretation is given in the paper. The first method was proposed by P. Cousot, the second one was developed on automation department in NSTU independent from the first one. Both methods base on common idea of source code representation to a model and following analysis of it. Both methods assume investigation on all possible paths of program execution. Examples of analysis of same program were presented in the paper. In addition, differences of considered methods were carried out in the paper. Specifically the abstract interpretation method assumes to use for automatically checking of one class errors with specifically used data domain. The second studied method allows to developer to execute arbitrary checks and errors that might be find with use of domain could be find in moment of program interpretation. Another different is that abstract interpretation mothed allows analyzing program's source code even without full information about system, despite the fact that precision will be lower. Full system representation required for the second method.

Keywords: software, testing, input intervals, formal verification, dynamic verification, verification, model checking, software models, graphs, total correctness of programs

REFERENCES

1. Cousot P., Cousot R. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. Conference record of the fourth ACM symposium on principles of programming languages, Los Angeles, California, Jan. 17–19, 1977. New York, ACM Press, 1977, pp. 238–252.

2. Voevoda A.A., Romannikov D.O., Zimaev I.V. Primenenie UML diagramm i setei Petri pri razrabotke vstraivaemogo programmnogo obespecheniya [An approach to the using UML and Petri nets for embedded software designing]. *Nauchnyi vestnik NGTU – Science bulletin of Novosibirsk state technical university*, 2009, no. 4 (37), pp. 169–174.

3. Clarke E.M., Grumberg O., Peled D. *Model checking*. Cambridge, London, MIT Press, 2001. (Russ. ed.: Klark E., Gramberg O., Peled D. *Verifikatsiya modelei programm: Model checking*. Moscow, MTsNMO Publ., 2002. 416 p.).

* *Received 8 august 2014.*

4. Bush W., Pincus J., Sielaff D. A static analyzer for finding dynamic progamming errors. *Software: practice and experience*, 2000, vol. 30, iss. 7, pp. 775–802. doi: 10.1002/(SICI)1097-024X(200006)30:7<775::AID-SPE309>3.0.CO;2-H

5. Glukhikh M.I., Itsykson V.M., Tses'ko V.A. Ispol'zovanie zavisimostei dlya povysheniya tochnosti staticheskogo analiza programm [The use of dependencies for Improving the precision of program static analysis]. *Modelirovanie i analiz informatsionnykh sistem – Modeling and analisys of information systems*, 2011, vol. 18, no. 4, pp. 68–79.

6. Romannikov D.O. O poiske vkhodnykh intervalov [On the search for input intervals]. *Sbornik nauchnyh trudov NGTU – Transaction of scientific papers of Novosibirsk state technical university*, 2014, no. 1 (75), pp. 140–145.

7. Islam S., Krinke J., Binkley D., Harman M. Coherent clusters in source code. *Journal of systems and software*, 2014, vol. 88, pp. 1–24.

8. Markov A.V., Romannikov D.O. Algoritm translyatsii diagrammy aktivnosti v set' Petri [Algorithm of automatic conversion of the activity diagram into Petri-net structure formats]. *Doklady Akademii nauk vysshei shkoly Rossiiskoi Federatsii – Proceedings of the Russian higher school Academy of sciences*, 2014, no. 1 (22), pp. 104–112.

9. Romannikov D.O. Nakhozhdenie oshibok obrashcheniia k nesushchestvuiushchim elementam massiva na osnovanii rezul'tatov analiza seti Petri [Finding errors or nonexistent array’s elements based on the results of the analysis Petri nets]. *Sbornik nauchnyh trudov NGTU – Transaction of scientific papers of Novosibirsk state technical university*, 2012, no. 1 (67), pp. 115–120.

10. Voevoda A.A., Markov A.V. Romannikov D.O. Razrabotka programmnogo obespecheniia: proektirovanie s ispol'zova-niem UML diagramm i setei Petri na primere ASU TP vodonapornoi stantsii [Software development: software design using UML diagrams and PETRI nets for example automated process control system of pumping station]. *Trudy SPIIRAN – SPIIRAS proceedings*, 2014, iss. 3 (34), pp. 218–231.