Статический анализ ПО с помощью его промежуточных представлений и технологий с открытым кодом

Зубов М.В., Пустыгин А.Н., Старцев Е.В.

Челябинский Государственный Университет, p2008an@rambler.ru.

This paper presents current existing approaches of using programs's representations for static analysis. This analysis can help with understanding and extending open-source software. The most common approaches were implemented in prototypes with open technologies. Also class-level detailed representation was developed and implemented in prototype to prove the assumption of more effective analysis on less detailed representation.

Статический анализ исходного кода с использованием методов обработки текста может быть достаточно неэффективным. На практике инструменты, выполняющие статический анализ, обычно формируют промежуточные представления исходного кода. Самое распространенное решение - использование абстрактного дерева разбора (AST) [1]. Его можно получать различными способами, в том числе инструментами генерации парсеров, такими как ANTLR[2] (Checkstyle[3] для языка Java), либо специфичными инструментами, такими как API компилятора ROSE (Compass/ROSE)[4], поддерживающего языки C, C++ и Fortran. В проекте Pylint[5] (для языка Python) для генерации AST используется отдельный свободно-распространяемый модуль logilab-astng [6], который может быть использован независимо от Pylint. Были разработаны 2 прототипа генератора промежуточных представлений, основанных на AST: первый - на основе свободного модуля logilab-astng[6] для языка Python, второй - на основе компилятора javac, входящего в состав открытой реализации средств разработки на Java - OpenJDK[7].

Принципиально другой вариант промежуточного представления - реляционное представление исходного кода анализируемого ПО. Для получения такого представления по грамматике исходного языка подготавливается схема реляционной базы данных, далее созданная по этой схеме база данных заполняется в соответствии с конструкциями исходного кода. В результате, процесс анализа сводится к выполнению различных запросов к этой базе. В современных инструментах используются SQL-подобные языки QUEL[8] (в OMEGA[9] для Ada, C и Pascal) и .QL [10] (SemmleCode[11] для Java). В настоящее время ведется создание прототипа построителя реляционного промежуточного представления для языка Java.

Концепция реляционного представления расширяется с помощью логических языков (Prolog в ASTLOG[12], Datalog в CodeQuest[13]). Эти языки гораздо больше подходят для анализа структур, которыми представляется исходный код, например, транзитивных замыканий. Запросы к базе знаний транслируются в SQL и выполняются на конкретной базе данных, либо же база данных может быть представлена в виде базы знаний.

Отдельно от реляционного представления выделяется направление статического анализа, просто основанного на выполнении запросов к исходному коду. Обычно такой подход не накладывает ограничений на внутреннее представление. Оно может быть как реляционным в системе CodeQuest[13], так и AST в ASTLOG[12]. Некоторые заявляют о потенциально любом представлении (PQL[14]). В основном, в концепции запросов к исходному коду используют логические языки как наиболее удобные и нацеленные на извлечение знаний. Однако в пользу удобства и гибкости существуют анализаторы, выполняющие запросы на естественном языке (английском)[15].

Зачастую промежуточное представление определяется целью анализа, как в случае с концепцией запросов. Так в свободном проекте BLAST [16] программа представляется виде автомата потока управления (CFA), описывающего набор состояний процедур и переходы между ними. Далее в соответствии с алгоритмом “абстракция и уточнение по контрпримерам” создаются абстрактные деревья достижимости (ART), которые описывают возможные пути распространения программы с той долей абстракции, которая используется на очередной итерации алгоритма.

Независимо от различных представлений можно выделить модель анализа с помощью моделей отражения[17]. Она основана на сравнении высокоуровневой модели программы, которую программист мыслит себе в процессе разработки, и реальной, полученной в результате. Этот подход не требует конкретного представления кода, и иногда может быть выполнен с использованием текстовой обработки. Так, в частности, он был реализован в одном из первых опытов применения подхода для изучения подсистемы виртуальной памяти NetBSD[17]. На основе этого подхода с помощью собственного генератора промежуточного представления для

Python, использующего logilab-astng[6], была построена и проанализирована высокоуровневая модель проекта с открытым исходным кодом на Python.

Промежуточные представления могут строиться как одноязыковые, так и мультиязычные. Универсальные решения позволяют производить типовые процедуры анализа для различных языков, однако не все детали конкретного языка в этом случае могут быть представлены. В качестве прототипа универсального представления было реализовано представление уровня структуры классов для языков Python и Java на основе прототипов генерации частных представлений.

Помимо указанных можно предложить классификацию представлений по уровню детализации. Так в академическом проекте Bauhaus project[18] для языков Ada, C, C++, C# и Java имеются 2 представления - низкоуровневое InterMediate Language, описывающее конструкции языка на синтаксическом и семантическом уровнях, и Resource flow graphs, описывающее архитектурные особенности программной системы. В анализаторе, использующем PQL[14], выделяется 3 уровня анализа: модель глобального программного дизайна, модель структуры программы, модель детального программного дизайна. Для созданного прототипа универсального представления для Python и Java был выбран средний уровень детализации - уровень структуры и связи классов.

При работе с любым представлением всегда важно иметь возможность соотнести анализируемый объект или область программы с ее местом в исходном коде. Это достигается сохранением координат объектов в представлениях, как и было реализовано в универсальном представлении для Java и Python. Полученное высокоуровневое представление позволяет производить анализ на уровне классов и архитектуры проекта, что позволяет снизить накладные расходы на выполнение анализа по сравнению с более детализированным представлением. За счет универсальности категорий ООП, существующих во всех объектных языках[19], полученное представление является полным для своего уровня анализа и, в то же время, универсальным для объектно-ориентированных языков.

Литература

1. *Ахо А.В., Ульман Д.Д. / Компиляторы. Принципы, технологии и инструментарии. / М.: Вильямс, 2008г.*
2. *ANTLR Parser Generator / www.antlr.org*
3. *Checkstyle – Checkstyle 5.5 / checkstyle.sourceforge.net*
4. *ROSE / rosecompiler.org*
5. *Pylint (analyzes Python source code looking for bugs and signs of poor quality) / www.logilab.org/857*
6. *logilab-astng (Python Abstract Syntax Tree New Generation) / www.logilab.org/856*
7. *OpenJDK / openjdk.java.net*

8. *Ingres 10.0 QUEL Reference guide / Ingres corp. 2010, code.ingres.com/ingres/main/src/tools/techpub/pdf/QUELRef.pdf*
9. *M. Linton / Queries and Views of Programs Using a Relational Database System / www.eecs.berkeley.edu/Pubs/TechRpts/1983/5296.html*
10. *O. de Moor, E. Hajiyev, M. Verbaere / Object-oriented queries over software systems / ACM Press, 2007.*
11. *Enabling tools for your project | SemmleCode / semmle.com/solutions/enabling-tools-for-your-projects*
12. *R. Crew / ASTLOG: A Language for Examining Abstract Syntax Trees / Microsoft Research, 1997.*
13. *E. Hajiyev / CodeQuest - Source Code Querying with Datalog / St. Anne's College, Oxford University, 2005.*
14. *S. Jarzabek / Design of Flexible Static Program Analyzers with PQL / IEEE Transactions on software engineering, vol. 24, no. 3, march 1998.*
15. *M. Kinmming, M. Monperrus, M. Mezini / Quering Source Code with Natural Language / 26th IEEE/ACM International Conference On Automated Software Engineering (ASE'2011).*
16. *MTC (Models and Theory of Computation): BLAST Project / mtc.epfl.ch/software-tools/blast/index-epfl.php*
17. *G.C. Murphy, D. Notkin, K. Sullivan / Software Reflexion models: Bridging the gap between source and high-level models / Third ACM SIGSOFT Symposium on the Foundations of Software Engineering, pages 18–28, New York, ACM Press, 1995.*
18. *Project Bauhaus. Software Architecture, Software Reengineering, and Program Understanding / www.bauhaus-stuttgart.de/bauhaus/index-english.html*
19. *И. Грэхем / Объектно-ориентированные методы. Принципы и практика / М.: Вильямс, 2004 г.*