электроники. Сер. ЭВТ. Вып. 9, 1971. – С. 36-39.
4. Вальковский В.А., Котов В.Е., Милошко Й. Алгоритмы, математическое обеспечение и архитектура многопроцессорных вычислительных систем. М.: Наука, 1982. – 336 с.
5. Гасанов Э.Э., Кудрявцев В.Б. Теория хранения и поиска информации. Москва: Физмат-лит, 2002. – 288 с.
6. Решетников В. Н. Моделирование информационного поиска в информационно-поисковых системах // Кибернетика. № 5, 1079. – С. 129-132.
7. Ким Д.П. Методы поиска и преследования подвижных объектов. М.: Наука, 1989. – 336 с.
8. Кнут Д.Э. Искусство программирования для ЭВМ. Т. 3. Сортировка и поиск. М.: Мир, 1978. – 355 с.
9. Колмогоров А.Н., Успенский В.А. К определению алгоритма / Успехи математических наук. Т.13, № 4, 1958. – С. 3-28.
10. Гасанов Э.Э. О сложности поиска в базах данных // Искусственный интеллект. Межвузовский сборник трудов. Саратов: Изд. СГУ, 1993. – С. 41-56.
11. Носков В.Н. О сложности тестов, контролирующих работу тестов логических схем // Математические заметки. Т. 18, № 1, 1975. – С. 137-150.

OPTIMAL INFORMATION-GRAPH MODEL ORGANIZATION INCLUSIVE SEARCH IN DESCRIPTOR SEARCH ENGINES

Ljalin V.E., Maltzev S.A., Taranucha V.P., Shishov D.R.

This article describes how to create an optimal information-graph model of the organization of inclusive search in descriptor search engine to provide continuous backup information. The evaluation of inclusive search considered optimal information and graphs with nedrevovidnoy tree structure. A detailed algorithm for solving the problem of inclusive search by building information graph.

Keywords: *information graph, IG, including search and sets.*

Лялин Вадим Евгеньевич, д.т.н., д.э.н., профессор, Заслуженный изобретатель РФ, заведующий Кафедрой математических технологий в нефтегазовом машиностроении (МТНМ) Ижевского государственного технического университета им. М.Т. Калашникова (ИжГТУ). Тел. 8-912-767-30-00. Email: velyalin@mail.ru

Мальцев Сергей Андреевич, к.т.н., ведущий инженер ЗАО «Группа Компаний Старт» (г. Москва). Тел. 8-929-682-90-19. E-mail: rshavas@mail.ru

Тарануха Владимир Прокофьевич, к.т.н., доцент, заведующий Кафедрой конструирования радиоэлектронной аппаратуры ИжГТУ. Тел. 8-341-250-40-22. Email: kra_dept@istu.ru

Шишов Дмитрий Родионович, аспирант Кафедры МТНМ ИжГТУ. Тел. 8-912-447-51-40. E-mail: d1men27@mail.ru

УДК 681.3.06(07)

ПРОТОТИП УНИВЕРСАЛЬНОГО ПРОГРАММНОГО ИНСТРУМЕНТА ДЛЯ ПОСТРОЕНИЯ МАШИННОГО КОММЕНТАРИЯ В ВИДЕ ДИАГРАММЫ КЛАССОВ ИСХОДНЫХ ТЕКСТОВ ПРОГРАММ

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

В статье описывается использование промежуточных представлений исходного кода для выполнения статического анализа. Предлагается расширить использование универсальных и многоуровневых представлений в одном общем. Представлен прототип программного инструмента, генерирующий и обрабатывающий такое представление.

Ключевые слова: статический анализ, промежуточное представление, машинные комментарии, исходный код, диаграмма классов, Java, Python.

Введение

Статический анализ облегчает выполнение типовых задач разработки. Эти задачи могут быть связаны с поиском ошибок, улучшением качества кода, внутренним тестированием [1]. Перспективным направлением является извлечение информации по исходному коду (построение машинных комментариев [2]).

Промежуточные представления

Промежуточное представление (далее просто «представление») программного текста – это синтаксически и семантически эквивалентный исходному коду набор данных, над которым выполняется анализ [3]. Представление может быть выполнено в виде различных наборов данных. Самый простой и наиболее изученный в области компиляции вариант – абстрактное синтаксическое дерево AST, фактически, это результат синтаксического разбора. Более сложное, обеспечивающее увеличение скорости анализа – реляционная база данных. Однако затраты на получение такого представления растут. Один из наиболее эффективных видов представления – конечный автомат. Помимо AST в компиляции широкое применение нашло представление SSA (Single-State Assignment) и его развитие GSA [4].

Представления, описывающие только один входной язык, будем называть частными, а те, что пригодны для исходного кода на нескольких входных языках, – универсальными. К частным представлениям можно отнести уже упомянутое абстрактное синтаксическое дерево, описывающее исходный текст программы во всех деталях, свойственных конкретному языку программирования. К универсальным представлениям можно отнести SSA и GSA, с той оговоркой, что версии переменных будут описываться в терминах конкретного языка.

Использование универсальных представлений сокращает общие затраты при анализе, так как общее количество требуемых функциональных модулей представления и реализации цели анализа для представления снижается. Пусть предполагается выполнение анализа для N исходных текстов, каждый из которых написан на языке высокого уровня, не совпадающим с языком любого из других текстов, и M целей анализа каждого текста. Тогда общее количество независимых функциональных модулей при использовании универсального представления составит $N + M$. Если предположить, что для выполнения анализа используется хотя бы синтаксическое дерево разбора (или другое частное представление), полученное с помощью транслятора, то отказ от универсального представления приведет к необходимости применить $N \times M$ независимых функциональных модулей.

Уже для трех языков высокого уровня (например java, python, cpp) и двух целей анализа (например получение диаграммы классов и графа вызовов) появляется выигрыш по количеству независимых функциональных модулей в системе на основе универсального представления, так как $N + M = 5$, а $N \times M = 6$.

Из основного достоинства универсального представления следует и недостаток: универсальные представления не позволяют отразить все детали синтаксиса каждого из языков исходного текста программ.

Универсальные многоуровневые представления

Для выполнения анализа нам вовсе не обязательно иметь полную информацию обо всех деталях исходного текста. Самым простым уровнем является наиболее детализированный. Он полностью соответствует исходному коду. В различных проектах выделяют разное количество уровней. Так, например, в анализаторе PQL предлагается применять три уровня: модель глобального программного дизайна, модель структуры программы, модель детального программного дизайна [5]. А в Bauhaus Project имеется два уровня: низкоуровневый Inter Mediate Language, детализированный до операторов, и Resource Flow Graphs, описывающий архитектуру [6]. Нечто подобное наблюдается и в организации компилятора GCC, в котором используются представления GIMPLE, CFG(Control Flow Graph), RTL, помимо классического AST [7]. Как видно, различные подходы сходятся в том, что есть полностью соответствующее коду представление и описывающее глобальную структуру программы, архитектуру. Между этими двумя крайними уровнями можно выделить дополнительные уровни представлений. С точки зрения анализа основной интерес представляют структурные единицы используемой парадигмы программирования и языка.

Например, для объектно-ориентированных языков это может быть представление уровня классов, их иерархии и взаимосвязей (похоже на диаграмму классов UML [8]). Извлекая лучшее из представленных идей классификации представлений, можно сделать вывод, что универсальные представления могут быть эффективны на более высоких уровнях абстракции. Так, например, программная архитектура в целом мало зависит от операторных возможностей конкретного языка. В качестве примера возьмем рассмотренный уровень классового представления. Для всех объектно-ориентированных языков характерны общие черты при работе с классами. Именно они отражены в диаграмме UML, и именно они были рассмотрены.

Описание универсального классового представления

Для прототипной реализации были выбраны весьма разные языки: Java и Python, со своими особенностями, предлагаемыми для ООП. Самое главное их отличие в том, что Java – это язык со строгой типизацией данных, а Python – с динамической. Это накладывает некоторые ограничения на получение связей агрегирования и композиции, потому что типы полей в Python определяются на этапе исполнения, а не на этапе компиляции, то есть они не могут быть получены напрямую из исходного кода. Для упрощения получения прототипа откажемся от получения типов и построения связей агрегирования.

В Python присутствует множественное наследование, которого нет в Java. В Java есть интерфейсы, которых нет в Python, но в Java есть множественное наследование у интерфейсов, а также класс может реализовывать несколько интерфейсов. В данном случае самым простым решением будет приравнять интерфейсы к классам. Фактически так и есть, интерфейс – это абстрактный класс, который не содержит никаких реализаций. Таким образом, представление будет допускать множественное наследование для всех входных языков, даже для тех, где его нет. Отказываясь от интерфейсов, также приходится отказываться от отношений реализации. В Python'е такое отношение не может быть реализовано средствами языка так, чтобы оно отличалось от отношения обобщения. Однако оно может присутствовать на уровне логики. Для предлагаемого представления удобно использовать древовидные структуры. В качестве формата данных хорошо подойдет XML. В качестве основного тега выступает тег Class, описывающий отдельный класс, в качестве дочерних тегов используются теги: Attr – описывает поля класса, Method – описывает методы класса, Parent – описывает родителей класса.

Графическое представление XML-схемы [9] изображено на рис. 1. Полный текст схемы можно получить из git-репозитория [10].

Реализация прототипа

Реализованный прототип поддерживает два входных языка программирования – Java и Python. Для Python использовался генератор на основе библиотеки logilab-astng, что позволило сразу получать информацию о классах проекта. Для Java использовался генератор, использующий AST. Дерево разбора получалось из XML-документа, генерируемого доработанным компилятором Open JDK. Реализация прототипа выложена в git-репозитории на открытый ресурс GitHub под свободной лицензией.

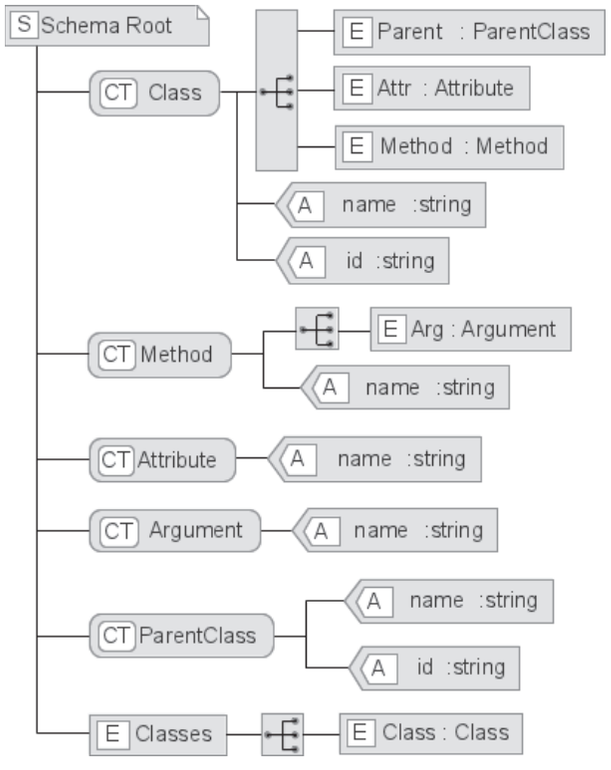

Рис. 1. Графическая XML-схема универсального классового представления

По мере реализации генераторов промежуточного представления для других объектно-ориентированных языков можно использовать уже написанные анализаторы, так как они не привязаны непосредственно к исходному тексту, а обрабатывают промежуточное представление

Использование построенного прототипа для визуализации иерархии классов

Для тестирования предложенного промежуточного представления была получена визуализированная иерархия классов в dot -формате описания графов из пакета Graphviz, так как иерархия наследования может быть представлена в виде направленного графа.

Для демонстрации универсальности представления и его полноты, независимо от входного языка, возьмем типовую реализацию шаблона проектирования «Посетитель» [11]. Были написаны две одинаковые по смыслу реализации на Java и Python. Полные тексты примеров доступны в git-репозитории [12].

Графические представления иерархии классов в UML-подобной форме для Python

на рис. 2, а для Java на рис. 3. Из рис. 2-3 видно, что несмотря на отличие стилей наименования классов и методов в Java и Python, диаграммы идентичны по смыслу.

Использование построенного прототипа для анализа иерархии классов

Объектная структура программного продукта в соответствии с современными представлениями должна удовлетворять определенным критериям, в том числе правильно построенным линиям наследования от базовых классов. С целью тестирования возможностей построенного прототипа было проверено выделение методов для групп объектов, которые могут быть вынесены в общий суперкласс.

Таблица 1. Результаты анализа Python-проектов

Проект	Всего методов	Проблемных методов
Django	2079	334
Twisted	8651	1291
Logilab	491	110

Таблица 2. Результаты анализа Java-проектов

Проект	Всего методов	Проблемных методов
Javac	2076	397
FindBugs	5538	787
SQuirreL	7798	1324

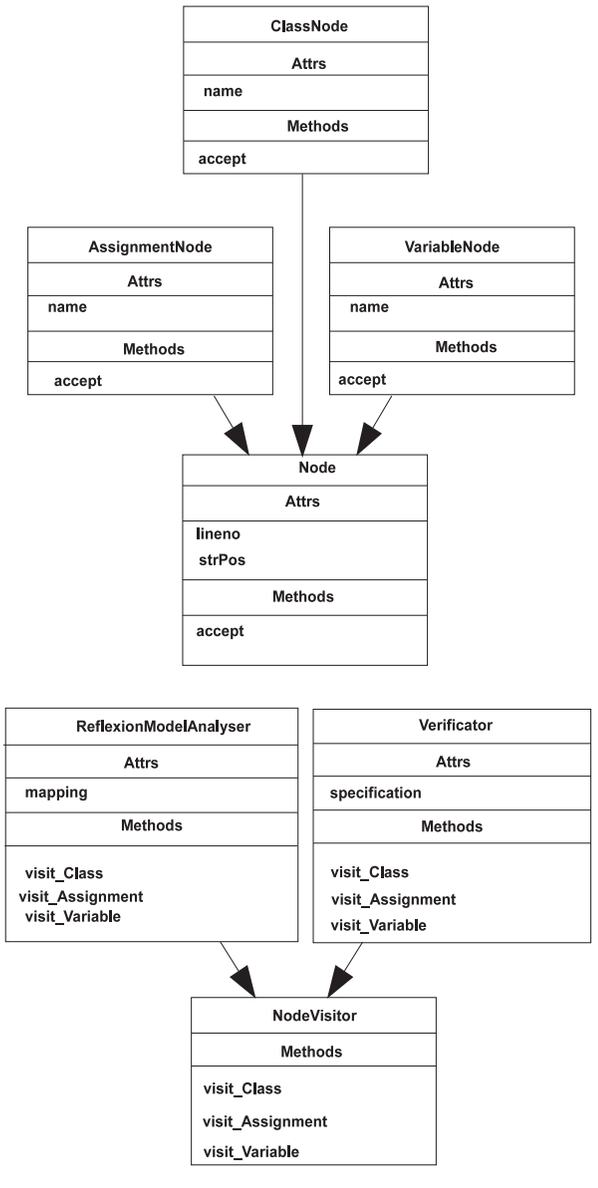

Рис. 2. UML-подобная диаграмма классов примера Python

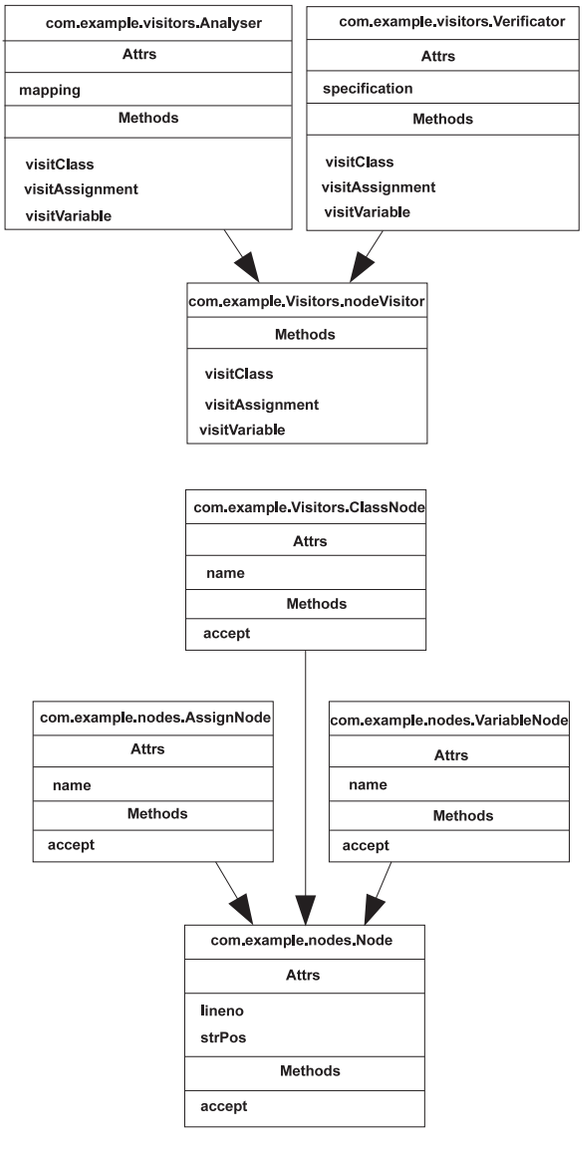

Рис. 3. UML-подобная диаграмма классов примера Java

Для анализа были выбраны несколько Python- и Java-проектов. На Python-библиотека для веб-разработки Django версии 1.4.1, сетевая библиотека Twisted версии 12.0.0, Python-пакет logilab, который использовался для обработки AST-деревьев (компоненты logilab-astng версии 0.23.1 и logilab-common версии 0.58.0). Проблемными методами названы те, чьи имена совпадают в разных классах, но не унаследованы ими от общего суперкласса. На Java анализировались компилятор javac, из состава OpenJDK v6u20, статический анализатор FindBugs v2.0.1, SQL-клиент SquirreL. Результаты анализа проектов представлены в таблицах 1-2.

Выводы

В результате проделанного исследования была разработана и описана модель промежуточного представления, позволяющая универсальным инструментом выполнять статический анализ исходного кода Java и Python. Представлен прототип такого инструмента. Для другого языка следует только получить универсальное промежуточное представление по предлагаемому шаблону.

Литература

1. Зубов М.В., Пустыгин А.Н., Старцев Е.В. Статический анализ ПО с помощью его промежуточных представлений и технологий с открытым исходным кодом. Львов: Львів, 2012. – С. 165-168.
2. Пустыгин А.Н., Иванов А.И., Язов Ю.К., Соловьев С.В. Автоматический синтез комментариев к программным кодам: перспективы развития и применения // Программная инженерия. №3, 2012. – С. 30-34.
3. Зубов М.В., Пустыгин А.Н., Старцев Е.В. Подходы к статическому анализу открытого исходного кода // Открытые технологии: Материалы VIII МК разработчиков и пользователей свободного программного обеспечения Linux Vacation / Eastern Europe 2012. Гродно, июнь 2012. Брест: Альтернатива, 2012. – С. 36-40.
4. Cook J., Gottlieb D., Greskamp B., Kujoth R. The Formation and Simulation of a «Whole Program» Gated Singular Assignment Program Dependence Graph. Unconventional Computer Architecture Group, Final Report. – 2008. // http://iacoma.cs.uiuc.edu/~greskamp/pdfs/497f pdf.
5. Jarzabek S. Design of Flexible Static Program Analyzers with PQL // IEEE Transactions on software engineering. Vol. 24. № 3, 1998. – P. 197-215.
6. Bauhaus project / mtc.epfl.ch/software-tools/blast/index-epfl.php
7. Basic gcc Intermediate Representation Dumps // http://www.cse.iitb.ac.in/~uday/courses/cs324-05/ gccProjects/node4.html

A PROTOTYPE OF UNIVERSAL SOFTWARE TOOL FOR BUILDING MACHINE COMMENT AS A CLASS DIAGRAM SOURCE CODE

Zubov M.V., Pustygin A.N., Startsev E.V.

This article describes using of source code intermediate representations for static analysis. It is proposed to extend using of generic and multi-level representations in one common. Software tool prototype that can make and process this kind of representation introduced.

Keywords: *static analysis, intermediate representation, machine comments, source code, class diagram, Java, Python.*

Зубов Максим Валерьевич, аспирант Челябинского государственного университета (ЧелГУ). Тел. 8-961-784-45-31. E-mai: zubovmv@gmail.com

Пустыгин Алексей Николаевич, к.т.н., доцент Кафедры компьютерной безопасности и прикладной алгебры ЧелГУ. Тел. 8-905-835-98-68. E-mai: p2008an@rambler.ru

Старцев Евгений Владимирович, аспирант ЧелГУ. Тел. 8-961-789-69-23. E-mail: slayer-gurgen@yandex.ru