УДК 004.054, 004.4'233

Медведев В.О., Рудаков И.В.
Московский государственный технический университет им. Н.Э. Баумана
(национальный исследовательский университет)

Верификация программного обеспечения формализованного сетью Петри

Аннотация: В данной работе представлено описание существующих видов статических анализаторов исходных текстов программного обеспечения; приведены преимущества и недостатки статических методов анализа. Приведено обоснование целесообразности использования статических анализаторов кода. Приведены особенности верификации программного обеспечения с помощью исполнимых моделей, описаны возможные формы представления этих моделей. Приведены примеры расширений конечных автоматов, которые могут быть использованы для описания исполнимых моделей. Описаны виды сетей Петри, выделены особенности верификации программного обеспечения с помощью иерархических сетей Петри.

Ключевые слова: Верификация, статический анализ кода, статическое тестирование, иерархическая сеть Петри, проверка исходных текстов программного обеспечения, исполнимая модель.

Для обеспечения корректности и надежности работы информационных систем используют различные методы верификации и валидации, позволяющие выявлять ошибки на разных этапах разработки и сопровождения программного обеспечения (ПО), чтобы устранять их. При этом важнейшим фактором при разработке программного обеспечения является конечная стоимость программного продукта. Согласно данным, приведенным в книге Макконнелла "Совершенный код" [1], пропорциональный коэффициент средней стоимости исправ-

ления дефектов ПО нелинейно увеличивается в зависимости от времени их устранения (таблица 1).

На основании этого широкое распространение получили средства верификации – статические анализаторы исходных текстов ПО, которые позволяют выявлять дефекты уже на этапе конструирования.

Таблица 1. Пропорциональный коэффициент средней стоимости исправления дефектов ПО в зависимости от времени их устранения (в условных единицах)

Время внесения дефекта	**Время обнаружения дефекта**				
	Выработка требований	**Проектирование архитектуры**	**Конструирование**	**Тестирование системы**	**После выпуска ПО**
Выработка требований	1	3	5 - 10	10	10 - 100
Проектирование архитектуры	-	1	10	15	25 - 100
Конструирование	-	-	1	10	10 - 25

Исторически статический анализ исходных текстов программного обеспечения (также называемый «статическое тестирование») принято разделять на 2 вида: рецензирование исходного кода и статический анализ кода.

Рецензирование исходного кода – один из самых старых и надежных методов выявления дефектов, который заключается в совместном внимательном чтении исходного кода и высказывании рекомендаций по его улучшению. В процессе чтения кода выявляются ошибки или участки кода, которые могут стать ошибочными в будущем.

К сожалению, рецензирование исходного кода имеет существенный недостаток – высокая цена. Для осуществления анализа кода этим методом необходимо регулярно привлекать команду опытных программистов. Также стоит отметить, что высокий объем кода значительно усложняет работу команды, что способствует увеличению сроков тестирования.

Со временем развитие технологий позволило частично автоматизировать рецензирование исходного кода и появился новый вид статического тестирования - статический анализ кода. В отличие от рецензирования, статический анализ благодаря автоматизации фактически не ограничен объёмом исследуемого кода. Тем не менее, результаты работы статического анализа всё же требуют изучения разработчиками для отделения реальных ошибок от неизбежных при таком подходе ложных срабатываний.

Статический анализ кода имеет следующие преимущества[2]:

– Полное покрытие кода - статические анализаторы проверяют даже те фрагменты кода, которые получают управление крайне редко. Такие участки кода, как правило, не удается протестировать другими методами.

– Не зависит от используемого компилятора - это позволяет находить скрытые ошибки неопределенного поведения, которые могут проявить себя при смене версии компилятора или версии операционной системы.

– Позволяет обнаруживать опечатки - нахождение этих ошибок другими способами является менее эффективным.

Также статический анализ кода имеет ряд недостатков[2]:

– Плохо справляется с диагностикой утечек памяти и параллельных ошибок, поскольку для их выявления фактически необходимо виртуально выполнить часть программы, что требует значительных вычислительных ресурсов и памяти.

– Ввиду универсального подхода к поиску дефектов, фактически осуществляется поиск «подозрительных» мест в исходных текстах. В результате этой особенности возможны ложно-позитивные срабатывания, поэтому результаты тестирования требуют проверки. Соотношение реальных дефектов к "шуму" может достигать один случай к ста на больших проектах.

Несмотря на очевидные недостатки, статический анализ исходных текстов позволяет значительно снизить стоимость ПО. Поэтому изначально простые инструменты статического анализа кода, использующие поиск по сигна-

турам, впоследствии были вытеснены анализаторами, проверяющими модель тестируемого ПО. Данная модель в зависимости от вида тестирования может быть представлена потоком управления, потоком данных и т.д. Основным достоинством этого типа анализаторов является меньше количество "шума" за счет частичного моделирования выполнения программ и возможность обнаружения более сложных дефектов.

Данный тип анализаторов требует получения различных моделей алгоритма – исполнимых моделей. Исполнимые модели (или операционные) характеризуются тем, что их можно каким-то образом выполнить, чтобы проследить изменение свойств моделируемого ПО. У такого подхода есть несколько особенно важных свойств [3]:

- В модели обычно учитываются не все свойства моделируемого ПО, а только важные для рассматриваемой в данный момент задачи.
- Модели, как правило, значительно проще моделируемых систем, поэтому их гораздо удобнее анализировать, а также можно выявить классы возможных при работе ПО проблем.
- За счет иного взгляда на систему часто можно увидеть такие ее характеристики и особенности, на которые ранее просто не обращали внимания.

Все виды исполнимых моделей можно считать расширением и обобщением конечных автоматов. Конечный автомат — это некоторая абстрактная модель, содержащая конечное число состояний чего-либо. Используется для представления и управления потоком выполнения каких-либо команд.

Существует достаточно большое количество обобщений и расширений конечных автоматов. В практике статического анализа кода применяют следующие виды:

1. **Системы помеченных переходов** - это тройка $D = \langle S, L, T \rangle$, где S - произвольное бесконечное множество, называемое множеством состояний, L - конечное множество меток (имен) переходов, $T \subseteq S \times L \times S$ - множество переходов.

2. **Расширенные конечные автоматы** – это тройка ⟨S, V, T⟩, где S – конечное множество состояний; V = I ∪ O ∪ R – конечное множество переменных, состоящее из входных сигналов (I), выходных сигналов (O) и внутренних регистров (R); T – конечное множество переходов.
3. **Взаимодействующие автоматы** – набор конечных автоматов, которые связаны каналами передачи реакций одного автомата как стимулов другого.
4. **Временные автоматы** – расширенные автоматы, содержащие набор дополнительных переменных-таймеров.
5. **Гибридные автоматы** – конченые автоматы, в которых часть переменных имеет значения, изменения которых обычно описывается системой дифференциальных уравнений.
6. **Сети Петри** – это тройка ⟨P, T, F⟩, где P – непустое множество позиций, T – непустое множество переходов, F ⊆ P×T ∪ T×P – отношение инцидентности. Сеть Петри определяется как двудольный граф [4]. Т.е. все вершины графа относятся к одному из двух классов – позициям и переходам. Позиции изображаются окружностями, переходы – отрезками прямой. Дуги в сетях Петри – направленные. Причем каждая дуга связывает вершины только разных классов. Сети Петри позволяют ввести состояния, внутри которых используются «фишки». Моделирование производится с помощью «запуска» сети – формально, фишки передвигаются по графу. Таким образом, существует понятие начальной маркировки сети Петри – расположение фишек внутри сети в начальный момент времени.

   Существует несколько видов сетей Петри:

– *Временная сеть Петри* — переходы обладают весом, определяющим продолжительность срабатывания (задержку).
– *Стохастическая сеть Петри* — задержки являются случайными величинами.

- *Функциональная сеть Петри* — задержки определяются как функции некоторых аргументов, например, количества меток в каких-либо позициях, состояния некоторых переходов.
- *Цветная сеть Петри* — метки могут быть различных типов, обозначаемых цветами, тип метки может быть использован как аргумент в функциональных сетях.
- *Ингибиторная сеть Петри* — возможны ингибиторные дуги, запрещающие срабатывания перехода, если во входной позиции, связанной с переходом ингибиторной дугой, находится метка.
- *Иерархическая сеть Петри* — содержит не мгновенные переходы, в которые вложены другие, возможно, также иерархические, сети. Срабатывание такого перехода характеризует выполнение полного жизненного цикла вложенной сети [5].

Статический анализ кода с помощью сети Петри имеют следующие преимущества:

- Сети Петри позволяют достаточно просто описать диаграммы потоков данных, причем параллелизм может быть описан явно.
- Проверка кода на наличие дефектов происходит достаточно просто и фактически заключается в поиске некорректной маркировки сети Петри.
- Иерархические сети Петри позволяют моделировать поведение вложенных функций и процедур.
- Сеть Петри может быть представлена как в табличном формате, так и в виде графа, что упрощает восприятие результатов статического анализа кода человеком.

Таким образом, одним из перспективных инструментов представления модели потоков данных является иерархическая сеть Петри. Параллелизм иерархических сетей Петри позволяет явно учитывать перегрузку функций и операторов, наследование классов, использование интерфейсов и т.п.

Ссылки на источники:

1. Макконнелл С. Совершенный код. Мастер-класс / Пер. с англ. — М. : Издательство «Русская редакция», 2010. — 896 стр. : ил. ISBN 978-5-7502-0064-1.
2. Статический анализ кода. [Электронный ресурс] Режим доступа: https://www.viva64.com/ru/t/0046/ (Дата обращения: 05.04.2017)
3. В.В. Кулямин. Методы верификации программного обеспечения. Институт системного программирования. Режим доступа: http://www.ict.edu.ru/ft/005645/62322e1-st09.pdf (Дата обращения: 25.02.2017)
4. Коротиков С. В. Применение сетей Петри в разработке программного обеспечения центров дистанционного контроля и управления: дис. канд. техн. наук. — Новосибирск: НГТУ, 2007.
5. Ломазова И.А. Вложенные сети Петри: моделирование и анализ распределенных систем с объектной структурой. – М.: Научный мир, 2004 – 208 с. ISBN 5-89176-247-1

Medvedev V.O, Rudakov I.V.
Bauman Moscow State Technical University

Software verification formalized by the Petri net

Annotation: In this paper, the existing types of static analyzers of source software texts are described; the advantages and disadvantages of static analysis methods are presented. The rationale of the static code analyzers usage is explained. Features of software verification using executable models are given; possible forms of representation of these models are described. Examples of extensions of finite automata that can be used to describe executable models are given. The types of Petri nets are described, features of software verification using hierarchical Petri nets are highlighted.

Keywords: verification, static code analysis, static testing, hierarchical Petri net, software source check, executable model