СТАТИЧЕСКИЙ АНАЛИЗ ИСХОДНОГО КОДА НА ЯЗЫКЕ TYPESCRIPT

К. А. Романчук

Белорусский государственный университет, г. Минск
e-mail: kostyaramanchuk@mail.ru
науч. рук. – А. В. Курочкин, ассистент

В работе представлены теоретические основы существующих подходов статического анализа программного обеспечения. Описан алгоритм построения абстрактного синтаксического дерева. Реализованы правила статического анализа для библиотеки tslint.

Ключевые слова: статический анализ кода; абстрактное синтаксическое дерево.

Одной из главных задач во время разработки программного обеспечения является своевременное обнаружение дефектов. Обнаружение и исправление ошибок в коде может занимать значительную часть от всего времени разработки ПО. Именно поэтому, автоматизация процесса обнаружения ошибок является актуальной задачей на сегодняшнее время. Составной частью этой автоматизации являются специальные статические анализаторы, в основе которых заложен метод статического анализа.

Статический анализ – это анализ, как правило, исходного кода программы, производимый (в отличии от динамического анализа) без реального её выполнения.

Существует 3 основных направления статического анализа исходного кода – синтаксический анализ, анализ потока данных и анализ потока управления.

Основной задачей синтаксического анализа является исследование одной или нескольких частей исходного кода без детального понимания контекста всей программы в целом. В ходе такого вида анализа идёт поиск конкретных или параметризируемых шаблонов. К примерам таких параметризируемых шаблонов можно отнести конструкции циклов или условий.

В отличии от синтаксического анализа, при анализе потока данных учитывается контекст применения той или иной конструкции. Виды анализа потока данных строятся на определении переменных в некоторой конструкции, значения которых используется без переопределения хотя бы на одном пути выполнения программы или переменных, которые имеют константные значения в конструкциях программы.

Анализ потока управления решает задачу определения свойств передачи управления между операторами программы. Конечным результа-

том анализа потока управления является построение графа потока управления – множество всевозможных путей выполнения программы. Далее происходит представление полученного графа в виде совокупности фрагментов определённого вида. Данное направление эффективно при решении задач оптимизации и преобразований программ.

Стандартным подходом для реализации статического анализатора является разбор исходного кода программы в абстрактное синтаксическое дерево, над которым осуществляется дальнейший анализ. Абстрактное синтаксическое дерево – конечное помеченное ориентированное дерево, в котором каждый узел соответствует очередному оператору в коде, а листья – операндам.

Для более глубоко понимания приведём пример построения абстрактного синтаксического дерева для небольшого фрагмента кода на TypeScript:

```
while (a < 10) {
  if (a > b) {
     console.log(a);
     a++;
  } else {
     Console.log(a + b);
     a++;
  }
}
```

На рис. 1 представлен разбор данного фрагмента в синтаксическое дерево.

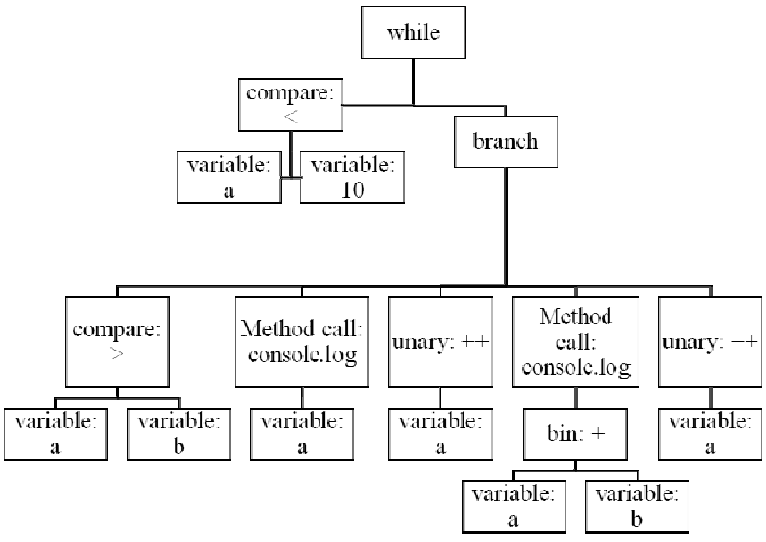

Рис. 1. Разбор фрагмента кода в абстрактное синтаксическое дерево

Статический анализатор, как правило, осуществляет обход этого дерева в глубину. В пределах одного поддерева при обходе анализатор может выделять некоторое временное состояние. Например, в рамках поддерева в виде стека может храниться информация о переменных, на-

ходящихся в области видимости. По мере объявления переменные добавляются на вершину стека, а при выходе из области видимости стек очищается.

Такое временное состояние позволяет осуществлять анализ поддерева в контексте проверки на корректное использования переменных. Так, например, если переменная, объявленная в области видимости некоторого поддерева исходного абстрактного синтаксического дерева, в дальнейшем не используется как операнд никакого другого оператора на любой глубине этого поддерева, можно сделать вывод о том, что такая переменная только объявляется, но в дальнейшем ни разу не используется в пределах своей области видимости, соответственно, такое объявление переменной может быть удалено.

Для изучения возможностей анализа абстрактного синтаксического дерева в рамках работы реализованы 2 правила для статического анализатора tslint – поиск бесконечных циклов и поиск «мертвого» кода по булевским литералам, являющимся тавтологиями.

Поиск бесконечных циклов осуществляет обход абстрактного синтаксического дерева в глубину с каждого узла, представляющего собой цикл с условным выходом, т.е. блоки конструкций while, do..while и for. К данной группе не относятся циклы for-of и for-in, т.к. в них условие выхода всегда одинаково и заключается в достижении последнего элемента коллекции или ключа структуры, по которой осуществляется итерация. Для конструкций while, do..while и for на этапе статического анализа может быть проанализировано используемое условие. С помощью логики упрощения булевых выражений можно определить, является ли логическое выражение, передаваемое в условие выхода из цикла, тавтологией (т.е. при любых значениях переменных эквивалентно true). Примерами тавтологий являются выражения true, x === x, x || !x и другие. Если условие выхода из цикла является тавтологией, цикл будет бесконечным в том случае, если внутри тела цикла не используется оператор break выхода из цикла или оператор return возврата из текущей функции. Таким образом, обнаружение бесконечного цикла осуществляется по одновременному выполнению всех 3 условий: поддерево является циклом while, do..while или for, условие выхода из цикла является тавтологией, в теле цикла отсутсвует использование операторов break и return. Наличие бесконечного цикла, скорее всего, является ошибкой, допущенной программистом при написании кода. Статический анализ таких участков может использоваться для указания на факт наличия потенциально некорректного кода.

Поиск «мертвого» кода осуществляется при помощи анализа потока управления. «Мертвый» код, т.е. код, который никогда не будет выпол-

няться, имеет место в тех случаях, когда до его выполнения при любых значениях переменных будет использован оператор, прерывающий выполнение текущего блока. Кроме того, в качестве «мертвого» кода могут выступать тела циклов и ветви условных операторов if/else в случаях, когда условие в них эквивалентно true или false вне зависимости от значений переменных, входящих в это условие. Например, если условие в ветви if является тавтологией, ветвь else будет являться «мёртвым» кодом. Как и в случае с бесконечными циклами, наличие «мертвого» кода в приложении может указывать на допущенную при написании кода ошибку. Однако, в некоторых случаях, «мёртвый» код оставляется намеренно, с целью вернуться к нему в дальнейшей разработке. В таких случаях статический анализ можно использовать при транспиляции для исключения таких участков из результирующего приложения.

Таким образом, статический анализ кода является важнейшей частью современных подходов к программированию. Статический анализ может использоваться как для поиска ошибок в коде на самом раннем этапе для помощи программисту, так и для более сложных видов анализа (например, анализа потока управления), на основании которых могут осуществляться оптимизации на этапе транспиляции или компиляции кода. Для практического рассмотрения существующих подходов к статической типизации были реализованы 2 правила для библиотеки tslint статического анализа для языка программирования TypeScript. Разработанные правила позволяют обнаруживать бесконечные циклы и «мертвый» код в пределах одной функции, и могут использоваться как для предупреждения ошибок при написании кода, так и для его оптимизации в процессе сборки.

Библиографические ссылки

1. *Nathan Rozenthals.* Mastering TypeScript, 2nd edition. Packt Publishing, 2017 – 364 p.
2. *Brian Chess, Jacob West.* Secure Programming with Static Analysis. Pearson Education, 2007. – 624 p.
3. *Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman*. Compilers: Principles, Techniques, and Tools, 2nd edition – MIT Press, 2006. – 1000 p.