А. В. Азаренко

МЕТОДЫ СТАТИЧЕСКОГО АНАЛИЗА ПРОГРАММНОГО КОДА

Рассматривается возможность оптимизации и повышения защищенности приложений посредством статического анализа программного кода.

Введение

Статический анализ кода это процесс выявления ошибок и недочетов в исходном коде программ. Статический анализ можно рассматривать как автоматизированный процесс обзора кода. Существенный недостаток методологии мануального обзора кода, это крайне высокая цена, поэтому компромиссным решением являются инструменты статического анализа кода, которые постоянно обрабатывают исходные тексты программ и выдают программисту рекомендации обратить повышенное внимание на определенные участки кода.

I. Основные задачи и приемущества

В целом задачи, решаемые с помощью методов статического анализа кода можно разделить на три категории:

1. Выявление ошибок в программном коде;
2. Рекомендации по оформлению программного кода. Некоторые платформы статического анализа позволяют проверять программный код на соответсвие заданному стандарту оформления программного кода;
3. Подсчет метрик, позволяющих получить численное значение того или иного свойства программного обеспечения.

Главное преимущество статического анализ программного кода состоит в возможности существенного снижения стоимости устранения дефектов в программном обеспечении. Чем раньше ошибка выявлена, тем меньше стоимость ее исправления.

Применение методов статического анализа программного кода позволяет выявить большое число ошибок на этапе конструирования, или написания программного кода, что существенно снижает стоимость разработки всего проекта.

Статический анализ программного кода обладает рядом других приемуществ. Одним из них является полное покрытие кода, что позволяет находить дефекты и уязвимости в обработчиках редких ситуацию, мануальное тестирование которых затруднено по тем или иным причинам. Статический анализ является независимым от среды использования и комплятора, что в свою очередь позволяет обнаруживать ошибки, которые могут проявиться в процессе длительного использования ПО.

II. Обзор методов на примере интервального анализа

Интервальный анализ вычисляет для каждой целочисленной переменной верхнюю и нижнюю границы для ее возможных значений. Интервалы являются интересными результатами анализа, поскольку эти результаты могут использоваться для оптимизации и обнаружения ошибок, связанных с проверкой границ массива, числовых переполнений и представления целых чисел. Данный случай включает решётку бесконечной высоты, и мы должны использовать специальный метод для обеспечения сходимости к фиксированной точке [1].

Решетка, описывающая одно абстрактное значение, определяется слудующим образом:

$$Intervals = lift(\{[l,h] \mid l,h \in N \wedge\ l \leq h\}), \quad (1)$$

где

$$N = \{-\infty, ..., -2, -1, 0, 1, 2, ..., +\infty\} \quad (2)$$

это множество целых чисел с бесконечным количеством конечных точек и порядком интервалов, определенных путем включения:

$$[l_1, h_1] \sqsubseteq [l_2, h_2] \Leftrightarrow l_2 \leq l_1 \wedge h_1 \leq h_2 \quad (3)$$

Эта решётка не имеет конечной высоты, поскольку она содержит, к примеру, следующую бесконечную цепь:

$$[0,0] \sqsubseteq [0,1] \sqsubseteq [0,2] \sqsubseteq [0,3] \sqsubseteq [0,4] \sqsubseteq [0,5]... \quad (4)$$

Это переносится на решетку абстрактных состояний

$$States = Vars \to Intervals \quad (5)$$

До указания правил ограничения необходимо определить фукцию eval, которая выполняет абстрактную оценку выражений:

$$eval(\sigma, X) = \sigma(X) \quad (6)$$

$$eval(\sigma, I) = [I, I] \quad (7)$$

$$eval(\sigma, input) = [-\infty, +\infty] \quad (8)$$

$$\begin{aligned} &eval(\sigma, E_1\ op\ E_2) \\ &= \widehat{op}(eval(\sigma, E_1), eval(\sigma, E_2)) \end{aligned} \quad (9)$$

Абстрактные арифметические операторы определяются как:

$$\widehat{op}([l_1,h_1],[l_2,h_2]) = [\min_{x\in[l_1,h_1],y\in[l_2,h_2]} x \; op \; y \; \max_{x\in[l_1,h_1],y\in[l_2,h_2]} x \; op \; y] \tag{10}$$

Например:

$$\widehat{+}([1,10],[-5,7]) = [1-5, 10+7] = [-4,17] \tag{11}$$

Функция JOIN является обычной для форвордного анализа:

$$JOIN(v) = \coprod_{w\in pred(v)} [\![w]\!] \tag{12}$$

Теперь мы можем указать правило ограничения для присвоений:

$$X = E: \; [\![v]\!] = JOIN(v)[X \mapsto eval(JOIN(v), E)] \tag{13}$$

Для остальных узлов ограничение является тривиальным:

$$[\![v]\!] = JOIN(v) \tag{14}$$

Интервальный анализ решётки имеет бесконечную высоту, так что применение алгоритмов фиксации точки может никогда не заканчиваться: для решётки Ln последовательность приближений

$$f^i(\bot, ..., \bot) \tag{15}$$

никогда не должна сходиться. Мощным инструментом для обхода подобного рода задачи является применение методов расширения и сужения.

III. Выводы

Существует множество методов статического анализа программного кода, однако ни один из них не является ультимативным, поэтому выбор того или иного метода статического анализа программного кода зависит от большого числа факторов таких, как среда разработки и применения, архитектурные особенности продукта и т.д.. Метод интервального анализа используется для решения проблемы неопределенного потока мощности в централизованном кластере, так что в интервал включаются все решения, что в свою очередь позволяет покрыть все возможные варианты использования.

1. Андрес Меллер и Махаэль И. Шварцбах, Статичтический анализ программ / А. Меллер, М. И. Шварцбах // 2019. – С. 75-77.

Азаренко Алексей Васильевич, магистрант кафедры информационных технологий и управления БГУИР, lehaazarenko@mail.ru.

Научный руководитель: Навроцкий Анатолий Александрович, заведующий кафедрой информационных технологий автоматизированных систем БГУИР, кандидат физико-математических наук, доцент, navrotsky@bsuir.by.