ИТМО

Программное обеспечение высоконагруженных систем

Можаев Андрей Михайлович

Выпускная квалификационная работа

Оптимизация строковой решетки для статического анализа кода на базе абстрактной интерпретации

Уровень образования: магистратура

Научный руководитель:
к.ф.-м.н. Дмитрий Сергеевич Шалымов

Рецензент:
зав. лаб. 37 в ИПУ РАН,
д.т.н Антон Викторович Уткин,

Санкт-Петербург
2025 г.

Содержание

Введение

Современное программное обеспечение становится все более сложным и масштабным, что увеличивает вероятность ошибок, возникающих в процессе разработки. Даже незначительные на первый взгляд ошибки могут привести к серьезным последствиям — от сбоев в пользовательских приложениях до срывов критически важных систем, таких как медицинское оборудование или системы управления транспортом. Именно поэтому проблема обеспечения надежности программных систем стоит особенно остро. Один из эффективных способов борьбы с потенциальными ошибками — применение методов статического анализа кода.

Статический анализ позволяет обнаружить широкий спектр ошибок и уязвимостей на ранней стадии разработки, еще до выполнения программы. В отличие от динамического анализа, который исследует поведение программы во время исполнения на различных входных данных, статический анализ работает только с исходным кодом или промежуточным представлением программы. Это дает возможность проверять весь возможный спектр входных данных и состояний программы, выявляя проблемы, которые могли бы проявиться лишь в редких и трудновоспроизводимых сценариях выполнения.

Одним из наиболее теоретически обоснованных и мощных подходов к статическому анализу является метод абстрактной интерпретации. Этот метод основан на идее упрощения множества всех возможных состояний программы путем перехода от конкретных значений переменных к их абстрактным представлениям в специально сконструированном математическом пространстве — абстрактном домене. В таком представлении каждое абстрактное значение описывает сразу множество возможных конкретных значений, а операции над абстрактными элементами являются оверапроксимацией (over-approximation) реального поведения программы.

Оверапроксимация означает, что абстрактный анализ учитывает не только те сценарии, которые реально могут возникнуть, но и теоретически возможные, но недостижимые состояния. Это свойство важно, поскольку оно позволяет гарантировать безопасность анализа: если статический анализ на

основе абстрактной интерпретации не обнаружил ошибок, то можно быть уверенным, что в реальном исполнении программы аналогичных ошибок действительно не произойдет — ни при каких входных данных и сценариях выполнения. Такой подход позволяет обнаруживать целые классы потенциальных ошибок, включая обращения к неинициализированным переменным, выходы за границы массивов, деления на ноль и другие критические дефекты.

Неподвижные точки и решетки

Фундаментальным математическим понятием в абстрактной интерпретации являются неподвижные точки. При выполнении статического анализа программа рассматривается как множество состояний, которые изменяются при выполнении инструкций. Однако особую сложность представляет анализ циклов и рекурсий, поскольку количество итераций заранее неизвестно и потенциально бесконечно. Полный перебор всех возможных состояний и проходов цикла невозможен на практике из-за комбинаторного взрыва.

Именно здесь применяется вычисление неподвижной точки — состояния программы, при котором дальнейшее выполнение цикла не приводит к появлению новых абстрактных состояний. В контексте абстрактной интерпретации это означает нахождение верхней границы множества возможных значений, которые могут принимать переменные после выполнения неопределенного числа итераций цикла. Такой подход позволяет "замкнуть"цикл, остановив анализ в тот момент, когда достигнута стабильность (фиксация значений) в абстрактном пространстве.

Чтобы анализ завершался за конечное время, применяется механизм ускорения сходимости — так называемый widening (расширение), который грубо обобщает накопленные результаты и ускоряет достижение неподвижной точки, жертвуя при этом частью точности ради производительности.

Вся эта процедура возможна благодаря тому, что абстрактные значения организованы в решетку — математическую структуру, определяющую отношения между абстрактными элементами и операции над ними. Решетка — это частично упорядоченное множество, в котором для любых двух элементов можно определить наименьшую общую верхнюю границу (join) и наиболь-

шую общую нижнюю границу (meet). Эти операции позволяют объединять и пересекать абстрактные состояния, обобщать или уточнять информацию в процессе анализа.

Примером простой решетки является множество булевых значений $\bot$, true, false, $\top$, где $\bot$ — невозможное состояние, $\top$ — любое, а операции join и meet позволяют вычислять общие свойства логических выражений.

Цели и задачи

Основной проблемой является недостаточная точность строковых абстрактных доменов. В некоторых случаях анализ не может корректно определить недостижимость определенных веток исполнения кода, что приводит к ложным срабатываниям и увеличению количества потенциальных ложных ошибок. Это снижает полезность статического анализа, так как разработчики вынуждены разбираться с ложными предупреждениями.

Целью данной работы является оптимизация строковой решетки, используемой в статическом анализе кода на базе абстрактной интерпретации, с целью повышения ее точности и эффективности.

Для достижения этой цели необходимо решить следующие задачи:

- Провести исследование существующих методов анализа строк в статическом анализе и выявить их недостатки. В частности, определить, в каких случаях текущие решетки недостаточно точно моделируют поведение строковых данных, приводя к ложным срабатываниям.
- Рассмотреть существующие подходы к построению строковых абстрактных доменов.
- Разработать и предложить оптимизированный вариант строковой решетки, сочетающий точность анализа и разумное время выполнения.
- Оценить предложенное решение экспериментально, сравнив его с существующими методами по критериям точности, производительности и масштабирумости.

Результатом работы станет усовершенствованная строковая решетка, позволяющая улучшить статический анализ кода за счет более точного представления строковых данных, сокращения ложных срабатываний и повышения эффективности вычислений.

Обзор предметной области

Статический анализ

Статический анализ кода — это метод выявления ошибок и уязвимостей без выполнения программы, работающий с исходным кодом или промежуточным представлением [1].

Ключевые преимущества

- **Полнота**: Анализирует все возможные пути выполнения, включая редкие сценарии
- **Раннее обнаружение**: Позволяет находить ошибки на этапе разработки
- **Безопасность**: Не требует выполнения потенциально опасного кода

Сравнение с динамическим анализом

- **Покрытие**: Статический — все пути, динамический — только выполняемые
- **Ресурсы**: Статический требует меньше вычислительной мощности
- **Точность**: Динамический анализ дает меньше ложных срабатываний

```
// Статический анализ обнаружит уязвимость:
String query = "SELECT * FROM users WHERE id = "+ userInput;
// Потенциальная SQL-инъекция

// Динамический анализ потребует:
// 1. Запуска программы
// 2. Подачи вредоносного ввода
// 3. Мониторинга поведения
```

Основные методы статического анализа

Символьное исполнение

Символьное исполнение — это метод статического анализа, при котором программа выполняется с абстрактными символьными значениями вместо конкретных данных. Каждая переменная представляется как символьное выражение (например, α, β), а условия ветвления записываются как логические формулы [2].

Принцип работы

- **Символьные переменные**: Вместо конкретных значений используются символы (например, `x` = α, `y` = β)
- **Путевые условия (path constraints)**: Для каждого пути исполнения строится логическая формула
- **SMT-решатель**: Проверяет выполнимость условий (используются Z3 [6], CVC5 [7] и др.)

Преимущества

- **Точность**: Может находить конкретные входные данные, приводящие к ошибке
- **Глубина анализа**: Выявляет сложные взаимосвязи между переменными
- **Поддержка строк**: Эффективен для анализа SQL-инъекций, XSS и др.

Недостатки и ограничения

- **Проблема циклов**: Для каждого числа итераций создается новый путь

```
for (int i = 0; i < n; i++)
// Проблема: n - символьное значение
// Экспоненциальный рост числа путей
```

- **Вычислительная сложность**: Требует решения NP-полных задач
- **Ограничения SMT**: Строковые теории часто неполны [5]
- **Проблемы с памятью**: Символьные структуры данных сложны для анализа

Оптимизации

- **Выборочная символизация**: Только для критических переменных
- **Гибридные подходы**: Комбинация с абстрактной интерпретацией
- **Эвристики для циклов**: Ограничение глубины анализа

Абстрактная интерпретация

Сравнение подходов

Критерий	**Символьное исполнение**	**Абстрактная интерпретация**
Точность анализа	Высокая (работает с конкретными значениями)	Средняя (аппроксимация)
Производительность	Низкая (комбинаторный взрыв)	Высокая (аппроксимация состояний)
Обработка циклов	Проблематична (экспоненциальный рост путей)	Эффективна (через неподвижные точки)
Применимость к строкам	Точный анализ конкретных строк	Работа с абстрактными шаблонами
Ложные срабатывания	Редко	Часто (из-за оверапроксимации)

Строковые абстрактные домены

Выводы

Заключение

Благодарность

Список литературы

[1] Cousot P., Cousot R. *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints* // POPL. 1977. https://doi.org/10.1145/512950.512973

[2] King J.C. *Symbolic Execution and Program Testing* // Communications of the ACM. 1976. https://doi.org/10.1145/360248.360252

[3] Christensen A.S., Møller A. *Precise Analysis of String Expressions* // SAS. 2003. https://doi.org/10.1007/3-540-44898-5_10

[4] Arceri V. et al. *Abstract Domains for String Analysis* // ACM Computing Surveys. 2022. https://doi.org/10.1145/3494523

[5] Zheng Y. et al. "SMT-Based String Analysis for Vulnerability Detection 2021.

[6] de Moura L., Bjørner N. "Z3: An Efficient SMT Solver"// TACAS. 2008.

[7] Barbosa H. et al. "cvc5: A Versatile and Industrial-Strength SMT Solver"// TACAS. 2022.