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:

—

Рецензент:

—

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.

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. Деградация производительности не должна превышать 20%
- Оценить предложенное решение экспериментально, сравнив его с существующими методами по критериям точности, времени исполнения и потреблению памяти.

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

Обзор предметной области

Статический анализ

Статический анализ программного кода представляет собой метод исследования программ без их выполнения. Основная цель статического анализа — выявление потенциальных ошибок, уязвимостей и других характеристик программного обеспечения на основе его исходного кода или промежуточного представления [1]. Этот метод используется в компиляторах, инструментах проверки кода, системах анализа безопасности и других приложениях.

Существует несколько техник статического анализа, включая анализ потока данных, анализ потока управления, типизацию, проверку соответствия спецификациям и другие. В отличие от динамического анализа, который требует выполнения программы, статический анализ проводится на уровне исходного кода, байт-кода или абстрактного синтаксического дерева.

Основные применения

- Поиск уязвимостей (CWE, OWASP Top 10)
- Оптимизация кода
- Верификация свойств
- Генерация тестов

Два важных метода статического анализа — символьное исполнение и абстрактная интерпретация — позволяют анализировать поведение программ без их непосредственного запуска, но имеют разные подходы и цели.

Символьное исполнение

Символьное исполнение (Symbolic Execution) представляет собой метод анализа, при котором программа исполняется не на конкретных входных данных, а на символьных переменных [2]. В процессе исполнения строится множество путей выполнения программы, а также логические ограничения, наложенные на переменные. Этот метод используется, например, для генерации тестов с высоким покрытием и обнаружения уязвимостей.

Преимущества

- Позволяет находить ошибки, недоступные при обычном тестировании [3]
- Даёт точные результаты в пределах анализируемых путей

Ограничения

- Страдает от проблемы комбинаторного взрыва, так как количество путей выполнения растёт экспоненциально [4]
- Требует сложных SMT Solver-ов для обработки выражений (Z3 [5], CVC5 [6] и др.)

Абстрактная интерпретация

Абстрактная интерпретация (Abstract Interpretation) основана на создании приближённых представлений возможных состояний программы [7]. Вместо работы с конкретными или символьными значениями, этот метод использует абстрактные значения, которые обобщают множества возможных состояний.

Преимущества

- Эффективный анализ больших программ [8]
- Гарантированная завершаемость

Ограничения

- Ложные срабатывания [9]
- Сложность разработки точных абстракций

Сравнение методов

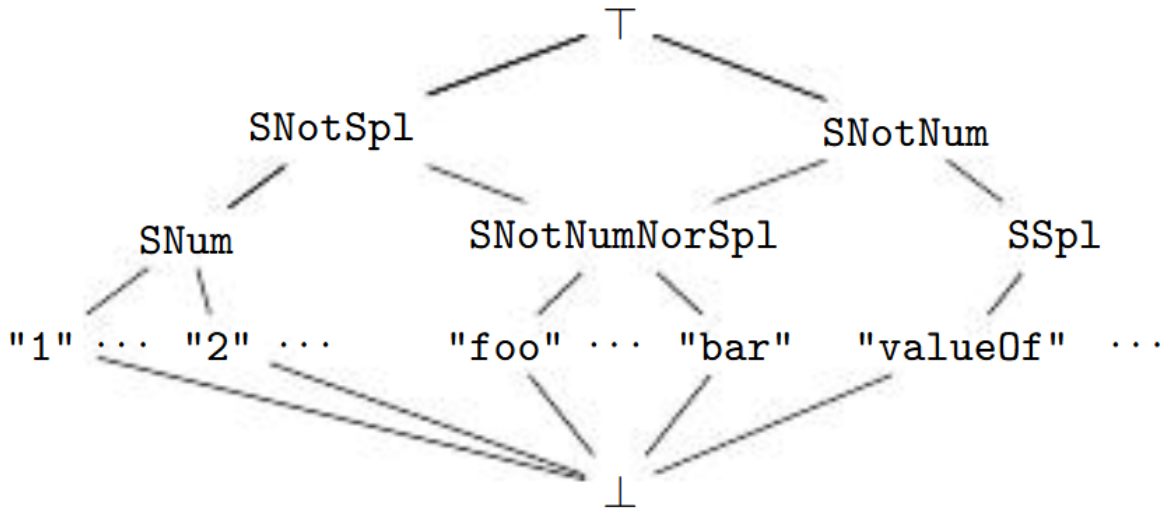
Характеристика	**Символьное исполнение**	**Абстрактная интерпретация**
Точность анализа	Высокая (по отдельным путям)	Приближённая (но глобальная)
Масштабируемость	Ограниченная (комбинаторный взрыв)	Высокая (благодаря абстракции)
Применимость	Поиск ошибок генерация тестов	Обнаружение уязвимостей верификация
Требования к вычислениям	Высокие (SAT solver)	Более низкие

Таким образом, оба метода находят своё применение в статическом анализе, а их комбинирование может дать более точные и эффективные результаты.

Строковые абстрактные домены

Пример простого строкового абстрактного домена реализован в системе **JSAI** — платформе статического анализа JavaScript [**?**]. В ней строки представляются в виде:

- любая ($\top$) и неинициализированная ($\bot$) строки
- константные строки
- категории “строка-число”, “спец-символ” и “любая строка”



Хотя этот подход обладает высокой производительностью, он страдает от недостатка точности. Например, пусть переменная s может быть равна строке “foo” или “bark”. Тогда значение выражения `s.length` может быть либо 3, либо 4, но абстрактный домен JSAI вернёт «любая длина». Это делает невозможным отбрасывание условий вроде `if (s.length == 0)` как недостижимых, что ведёт к ложным срабатываниям

Листинг 1: Пример недостаточной точности в строковом домене JSAI

```
let s;
if (input == 0) {
    s = "foo";
} else {
    s = "bark";
}
if (s.length == 0) {
    throw new Error("Divide by zero!");
}
let repeats = 100 / str.length;
```

Уточнение анализа с помощью конечных автоматов

Для повышения точности анализа был предложен подход, основанный на **конечных автоматах с переходами по буквам**, описанный в работе [**?**]. В этом подходе множество возможных значений строк моделируется как язык, распознаваемый конечным автоматом. Операции над строками соответствуют операциям над языками: объединение, пересечение, конкатенация и пр.

Преимущества

- высокая точность

Недостатки

- высокая вычислительная сложность
- большое потребление памяти
- низкая масштабируемость при анализе больших программ

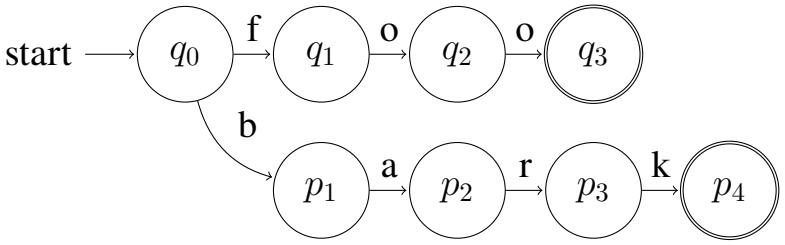


Рис. 1: Конечный автомат, распознающий строки “foo” и “bark”

Овер-аппроксимация строк в конечных автоматах повышает точность анализа строк во многих сценариях, но она не подходит для реальных программ, работающих со статически неизвестными входными данными и манипуляциями с длинным текстом

TARSIS

TARSIS (Template-based Abstract Representation of Strings with Static analysis) — это современный строковый абстрактный домен, предложенный в работе [**?**]. Основное новшество Tarsis заключается в том, что он работает с алфавитом строк, а не с отдельными символами. С одной стороны, такой подход требует более сложного и уточненного определения расширяющего оператора и абстрактной семантики строковых операторов. С другой стороны, это позволяет получать более точные результаты

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] Cadar C., Sen K. "Symbolic Execution for Software Testing"// IEEE Software. 2013.

[4] Baldoni R. et al. "A Survey of Symbolic Execution Techniques"// ACM Computing Surveys. 2018.

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

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

[7] Cousot P., Cousot R. "Abstract Interpretation Frameworks"// JLP. 1992.

[8] Blanchet B. et al. "A Static Analyzer for Large Safety-Critical Software"// PLDI. 2003.

[9] Cousot P. et al. "Why Does Astrée Scale?"// FMSD. 2012.

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

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

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