УДК 519.686.4
AMS MSC2020: 68N20

Исчисления алиасов для Си-подобных языков

Лыгин Л. И., Шилов Н. В.

Университет Иннополис

Аннотация. В нашей работе мы представляем реализацию двух вариантов исчисления алиасов (синонимичных адресов) для языка с адресной арифметикой — варианта, описанного в работе [2], и нового «легкого» варианта для обнаружения утечек памяти. Разработка наших вариантов исчисления была вдохновлена исчислением алиасов для языка без адресной арифметики из работы [1].

Ключевые слова: модели памяти, адресная арифметика, арифметика Пресбургера, проблема алиасинга, исчисление алиасов.

Введение

Несмотря на то, что бóльшая часть современного программного кода написана на языках с автоматическим управлением памятью и сборкой мусора, большáя часть критического кода по-прежнему написана на процедурных языках типа Си с прямым доступом к памяти, особенно в «жестких» средах, таких как микроконтроллеры или критичные к производительности центры обработки данных. Языки типа Си позволяют программисту обращаться к памяти напрямую практически по любому адресу, потому что все указатели на память являются просто целыми числами, и всегда можно привести любое целое число к указателю. Хотя это позволяет создавать эффективные приложения, это также опасно, потому что нет никаких барьеров для предотвращения несанкционированного доступа программы (например, к нераспределенной памяти) или у т е ч е к п а м я т и (например, выделение массивов с последующим «забыванием» о них), некоторые из которых могут привести к утечкам памяти в случае сбоя во время выполнения.

В некоторых случаях могут помочь автоматические тесты, но тестирование на утечки памяти (ситуации, которые возникают, когда программа выделяет часть памяти, а затем «теряет» все ссылки на нее и поэтому не имеет возможности когда-либо ее удалить), как известно, сложно. Гораздо лучше проверять использование памяти автоматически, не выполняя целевую программу, потому что это позволяет интегрировать готовые к использованию автоматизированные инструменты в интегрированные среды разработки (IDE) и конвейеры непрерывной интеграции (CI).

Проверка использования памяти при наличии адресной арифметики (то есть в ситуации, когда программа может попытаться получить доступ к памяти в любом «вычислимом» месте памяти с целочисленным адресом) может быть решена с помощью множества методов с разной степенью точности и эффективности. Одна конкретная проблема в этой области — анализ алиасинга, определяющий, могут ли два указателя (или должны, в зависимости от подхода) указывать на одно и то же место в «куче» (то есть — динамической памяти). Заметим, что проблемы с псевдонимами (алиасами — aliases) могут возникать даже без явного использования адресной арифметики, так как многие современные языки поддерживают указатели (или обычно используют только ссылки для хранения значений), а наличие поддержки для указателей подразумевает, что могут быть две переменные, которые «указывают» на одно и то же место в памяти.

1. Обзор содержания выполненной работы

В нашей работе мы представляем реализацию двух вариантов одного из методов статического анализа алиасинга — исходного (описанного в работе [2]) и нового «легкого» варианта исключительно для обнаружения утечек памяти. Исходный вариант исчисления был «вдохновлен» идеей исчисления алиасов без адресной арифметики из работы [1]. Вариант исчисления алиасов из [2] вычисляет по программе на модельном процедурном языке программирования MoRe (с адресной арифметикой) наборы синонимов и антонимов (то есть равенств $x = y$ и неравенств $x \neq y$ адресных выражений), а «легкий» вариант исчисления алиасов основан исключительно на равенствах адресных выражений. Мы также приводим результаты

экспериментов по проверке корректности и эффективности обоих вариантов исчисления для обнаруживая утечки памяти в процедурных программах.

Синтаксис языка MoRe задан следующим образом:
Определение 1.

$$P ::= \; skip \mid var\ V = C \mid V := T \mid$$
$$\mid V := cons(C^*) \mid [V] := V \mid V := [V] \mid dispose(V) \mid$$
$$\mid (P; P) \mid (if\ F\ then\ P\ else\ P) \mid (while\ F\ do\ P).$$

В языке MoRe есть только два типа данных — «числа» и адреса с возможностью неявно преобразовывать числа в адреса, а числовые выражения — в адресные выражения. Про тип адресов мы предполагаем, что теория первого порядка этого типа является разрешимой. Например, если множество адресов — это все натуральные числа, тогда адресная арифметики — это арифметика Пресбургера.

В качестве примера анализа, осуществимого с использованием обоих наших вариантов исчисления алиасов для языка MoRe, рассмотрим следующую программу
Пример 1. `var x = 0 ; x := cons(1, 3) ; x := 3`

Для этой программы анализ должен обнаружить (и обнаруживает) следующие утечки памяти:

- `«Memory leak - x is no longer reachable after assignment on statement x := 3 line 3»` — адрес первого элемента «двухэлементного списка», состоящего из 1, 3, более недоступен,
- `«Memory leak - x + 1 is no longer reachable after assignment on statement x := 3 line 3»` — адрес второго элемента «двухэлементного списка», состоящего из 1, 3, более недоступен,

из-за присваивания адресной переменной `x`, хранившей адрес начала списка, нового значения.

Заключение

Тестирование подтвердило перспективность нашего подхода для статического анализа утечек памяти, в то время как масштабируе-

мость и полезность для анализа промышленного кода нуждаются в дополнительных исследованиях.

На данный момент мы не провели сравнения нашего подхода с другими подходами к анализу алиасинга [3]. Такое сравнение поможет понять сильные и слабые стороны, а также предоставить полезную информацию о том, как можно улучшить наш анализ.

Другая важная задача — тестирование реализации на более крупных примерах. На данный момент все тестовые примеры содержат не более 20 строк кода, тестирование с использованием более крупных примеров может выявить серьезные узкие места в производительности.

Еще одна важная задача на будущее — «масштабируемость» теории, распространение подхода на языки с вызовами функций и объектно-ориентированные языки.

Список литературы

[1] *Meyer, B.* Steps towards a theory and calculus of aliasing // International Journal of Software and Informatics. — 2011. — Vol. 5. — P. 77–116.

[2] *Shilov, N. V.* Alias calculus for a simple imperative language with decidable pointer arithmetic / N. V. Shilov, A. Satekbayeva, A. P. Vorontsov // Bulletin of the Novosibirsk Computing Center. Computer Science series. — 2014. — №37. — P. 131–148.

[3] *Smaragdakis Y.* Pointer Analysis / Y. Smaragdakis, G. Balatsouras // Foundations and Trends in Programming Languages. — 2015. — Vol. 2, №1. — P. 1–69.

Библиографическая ссылка

Лыгин, Л. И. Исчисления алиасов для Си-подобных языков / Л. И. Лыгин, Н. В. Шилов // Всероссийская научная конференция «Математические основы информатики и информационно-коммуникационных систем». Сборник трудов. — Тверь : ТвГУ, 2021. — С. 199–203.
https://doi.org/10.26456/mfcsics-21-29

Сведения об авторах

1. Лыгин Леонид Ильич
Университет Иннополис. Студент

Россия, 420500, г. Иннополис ул. Университетская, д.1
E-mail: l.lygin@innopolis.ru

2. Шилов Николай Вячеславович
Университет Иннополис. Доцент

Россия, 420500, г. Иннополис ул. Университетская, д.1
E-mail: shiloviis@mail.ru