УДК 004.415.53

Yet Another Random Program Generator [1] - генератор случайных тестов для верификации оптимизаций в компиляторах языков C/C++

В. Ю. Ливинский[1], А. В. Митрохин[1], Д. Ю. Бабокин[2]

[1]Московский физико-технический институт (государственный университет)

[2]АО "Интел А/О"

В современном мире при разработке промышленных компиляторов возникает задача тестирования корректности работы оптимизаций. Общепринятые методы верификации зачастую обеспечивают недостаточное покрытие сложных компиляторных оптимизаций. Одним из путей улучшения качества тестирования является использование технологии рандомизированного тестирования. Она предполагает автоматическое создание огромного количества тестов и оценку корректности их работы.

Наиболее развитыми решениями для автоматического тестирования компиляторов языков C/C++ на текущий момент являются генераторы Csmith [2], Quest [3] и Orange [4]. Их отличительной особенностью является способность генерировать тесты, удовлетворяющие стандарту языка и в некоторых случаях нацеленные на конкретные компиляторные оптимизации, однако примененные в перечисленных выше генераторах методы не позволяют качественно верифицировать сложные оптимизирующие фазы [5]. При разработке подобных генераторов возникает две основные проблемы. Во-первых, необходимо гарантировать отсутствие неопределённого поведения (англ. undefined behavior) в генерируемых программах. Во-вторых, необходимо одновременно обеспечить разнообразие генерируемого кода и частое срабатывание оптимизаций.

Данная работа посвящена разработке генератора тестов, корректных с точки зрения стандарта, для верификации компиляторов языков C/C++. Его отличительной особенностью является поддержка генерации всего возможного спектра арифметических выражений, скалярных и цикловых участков кода, а также наличие механизмов, позволяющих целенаправленно тестировать арифметические и цикловые оптимизации.

Представленный генератор состоит из двух отдельных частей - генератора скалярных участков кода и генератора цикловых участков кода. В скалярных участках наибольшую сложность представляет работа с переменными и отслеживание отсутствия переполнений, в то время как в цикловых участках главной проблемой является контроль кросс-итерационных зависимостей и генерация разнообразных шаблонов доступа в память. В основе принципа работы обеих компонент лежит идея построения общей структуры и логики работы тестовой программы в высокоуровневом внутреннем представлении, не привязанной к конкретным выражениям или свойствам языка, и последующего перевода этого представления в исходный код теста. Данный подход позволяет контролировать значения всех переменных и свойств программы в каждой её точке и проводить эмуляцию исполнения теста во время его создания.

Генератор скалярных участков кода предназначен для порождения разнообразных преобразований переменных в теле теста посредством создания произвольных арифметических выражений и их комбинаций. При этом присутствует возможность задать преобладающие для региона кода операции, соотношение констант, индекс переиспользования переменных, общие подвыражения и другие параметры. Все это призвано повысить вероятность срабатывания различных арифметических оптимизаций. Используемые в тесте переменные могут иметь как целочисленный тип, так и быть структурами с битовыми полями и произвольной глубиной вложенности. Более того, поддерживается генерация различного ветвления потока управления, в том числе с произвольной глубиной вложенности условных операторов.

Генератор цикловых участков кода является важной частью генератора. Ключевым элементом генератора цикловых участков является его внутреннее представление массивов. Для каждого физически выделяемого в памяти массива создаётся один или несколько объектов с метаинформацией, в которой закодировано каким образом данный массив может быть

интерпретирован. Существует два возможных способа работать с массивом – работать с ним как со структурой массивов (*SoA*), или как с массивом структур (*AoS*). Первый шаблон нередко встречается в различных перестановочных алгоритмах, таких как алгоритмы шифрования и кодирования. Второй шаблон является простым обходом массива с некоторым шагом. Выбор указанных примитивов обусловлен в первую очередь необходимостью максимизировать вероятность векторизации кода, а также увеличить шансы на генерацию кода, наиболее часто встречающегося в современных программных продуктах. Нетрудно, однако, показать, что с помощью различных комбинаций шаблонов *AoS* и *SoA* можно добиться генерации цикла произвольной конфигурации и степени вложенности.

Для устранения неопределённого поведения в коде, создаваемом как генератором скалярных участков, так и генератором цикловых участков, был использован метод анализа небезопасных операций на этапе генерации и изменения или перестроения тестовой программы, предложенный в работе [4]. Изначально он был разработан для применения в скалярных участках кода, содержащих работу только с переменными целочисленных типов. Этот подход был расширен для применения на участках кода с произвольным ветвлением потока управления и цикловых участках. В данной работе был введен ряд ограничений, призванных упростить процесс разработки генератора и сделать процесс тестирования как можно более прозрачным и надёжным.

Во-первых, примененный алгоритм не поддерживает работу с числами с плавающей запятой. Выражения, содержащие такие числа, крайне трудно поддаются проверке на корректность из-за потери точности, связанной с округлением чисел и преобразованиями во время оптимизации кода в компиляторе.

Во-вторых, компонента генератора циклов поддерживает только массивы, содержащие одно значение во всех своих элементах. Это ограничение было продиктовано необходимостью отслеживать неопределённое поведение во всех арифметических выражениях. При этом достаточно выполнить внутри генератора всего одну итерацию цикла, чтобы сделать заключение о корректности кода. Необходимо заметить, что хотя интуитивно кажется, что подобный подход ограничивает выразительность генерируемых программ, с точки зрения компилятора это не совсем так. Входные массивы и большинство скалярных переменных инициализируются во внешнем файле, и у компилятора нет возможности узнать, что в массиве лежат одинаковые значения – эта информация доступна только генератору.

Для принятия решения о корректности исполнения был использован метод дифференциального тестирования, предложенный в работе [2]. В данной работе этот метод был расширен использованием санитайзеров Clang, которые служат для проверки отсутствия неопределённого поведения в генерируемых тестах.

Генератор, созданный в рамках данной работы, позволил обнаружить более 50 ошибок в GCC и Clang, а также сравнимое количество в проприетарных компиляторах. Эти ошибки были обнаружены в самых разнообразных компонентах компиляторов, что позволяет судить о высокой эффективности разработанного генератора и его способности качественно протестировать сложные оптимизирующие фазы.

Литература

[1] https://github.com/01org/yarpgen

[2] *Yang X., Chen Y., Eide E. and Regehr J.*: Finding and understanding bugs in C compilers. // Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation – 2011 – Pp. 283-294

[3] *Lindig C.*: Find a compiler bug in 5 minutes. // Proc. ACM International Symposium on Automated Analysis-Driven Debugging. – 2005 – Pp. 3-12.

[4] *Nagai E., Hashimoto A., Ishiura N.*: Reinforcing Random Testing of Arithmetic Optimization of C Compilers by Scaling up Size and Number of Expressions. // IPSJ Transactions on System LSI Design Methodology – 2014 – Vol. 7 – Pp. 91-100

[5] *Ливинский В.Ю., Митрохин А.В., Бабокин Д.Ю.*: Исследование методов автоматической генерации случайных программ для тестирования компиляторов языков C/C++. // 58-я научная конференция МФТИ - 2015