УДК 004.021

Студ. А.Н. Зайцев
Науч. рук. доц. А.А. Дятко
(кафедра информатики и веб-дизайна, БГТУ)

СТАТИЧЕСКИЙ АНАЛИЗ ЭФФЕКТИВНОСТИ ИСПОЛЬЗОВАНИЯ КОНТЕЙНЕРОВ ИЗ БИБЛИОТЕКИ STL В ПРОГРАММАХ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ C++

В современном мире наблюдается тенденция смещения акцента при выборе средств разработки к более высокоуровневым, которые сокращают время, затрачиваемое на разработку программного комплекса. Также развиваются различного вида «помощники» для программистов, которые облегчают различные стороны разработки: мощные интегрированные среды разработки, автоматическое форматирование кода, средства виртуализации и контейнеризации и т.д.

Одним из самых популярных методов автоматической проверки программ является статический анализ. Статический анализ — анализ программного обеспечения, производимый (в отличие от динамического анализа) без реального выполнения исследуемых программ [1]. Данный вид анализа позволяет предупредить множество потенциальных проблем с программой: использование освобождённой памяти, неоптимальное использование языковых средств и множество других проблем. В ходе данной работы мною был реализован статический анализатор, который позволяет определить, оптимален ли контейнер для работы с данными, которые выбирает программист в тех или иных ситуациях в программах на языке программирования C++.

Для реализации данного проекта мною был использован набор LLVM (Low-Level Virtual Machine) — набор библиотек для разработки компиляторов и утилит для работы с программами [2]. Из LLVM для реализации проекта я использую компилятор Clang [3] и статические анализаторы Clang Static Analyzer[4] и Clang Tidy[5].

В данной работе анализируется использование следующих контейнеров и Стандартной Библиотеки Шаблонов (Standard Template Library, STL): std::vector, std::list, std::forward_list, std::array, std::stack, std::queue, std::deque.

В связи со сложностью задачи введены следующие ограничения:

- анализ проводится только на уровне функций. Не допускается анализ между функциями и между разными единицами трансляции;
- анализ ограничен только подмножеством STL из-за сложности поддержки большого множества контейнеров;

- не поддерживается автоматическая замена контейнеров в исходном коде программы ввиду ограничений выбранного для данной работы статического анализатора и сложности сохранения корректности при такой замене.

В данном статическом анализаторе используется анализ абстрактного синтаксического дерева (Abstract Syntax Tree, AST). Если говорить более конкретно, то используется сопоставление с паттерном по AST.

В ходе анализа нас интересуют следующие метрики для каждого контейнера:

- количество операций поиска в контейнере;
- количество операций удаления элемента из контейнера;
- количество операций добавления элемента в контейнер;
- известен ли начальный и/или конечный размер контейнера при его создании.

Ниже на рис. 1 представлена схема работы данного статического анализатора.

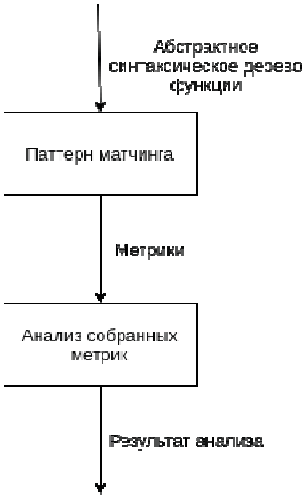

Рисунок 1 – Схема работы данного статического анализатора

Для сбора метрик для каждого контейнера были написаны специальные поисковые алгоритмы на специальном встраиваемом языке из CSA (embedded Domain Specific Language, eDSL).

Затем на AST каждой функции запускается механизм, который сопоставляет различные части AST с паттернами, которые были описаны выше. Если было обнаружено какое-то совпадение, то AST кода сохраняется в кеш. После того начинается стадия обработки собранных данных. Оно производится с помощью обыкновенного дерева различных вариантов. В конце работы данного алгоритма исходя из полученных метрик выносится решение о целесообразности применения в данном месте того или иного контейнера. Данный анализатор обладает следующими недостатками:

- недостаток информации о паттерне исполнения программы, что может крайне негативно сказываться на качестве проводимого анализатора;
- не используются анализ потока графа управления и метод символьного исполнения.

Возможные пути улучшения качества анализа:

- использование динамического анализа (например, PGO [6]) для получения более точных метрик о паттерне поведения программы.

Это в том числе поможет более точно определять «горячие» и «холодные» пути выполнения программы;

- проведение анализа не только для контейнеров из библиотеки STL. Это сделать достаточно сложно, так как нужна информация о том, как контейнер себя ведёт. Этого можно добиться с помощью ручного аннотирования контейнеров и проведения различных бенчмарков с сохранением результата в базе данных статического анализатора.

Похожие исследования в данной области проводили следующие два проекта:

- Cozy [7]. Проект по генерации контейнеров из высокоуровневого описания требований к контейнеру. Открытая реализация;
- Chameleon [8]. Проект для языка программирования Java по выбору оптимального контейнера в зависимости от текущей нагрузки и замена «на лету» контейнера при помощи модификации JVM.

С предварительной реализацией данного статического анализатора можно ознакомиться по следующей ссылке: https://github.com/ZaMaZaN4iK/llvm-project

ЛИТЕРАТУРА

1. Wikipedia : Staticprogramanalysis [Электронный ресурс] / https://en.wikipedia.org/wiki/Static_program_analysis
2. TheLLVMCompilerInfrastructure [Электронный ресурс] / https://llvm.org
3. Clang: a C language family frontend for LLVM [Электронный ресурс] / https://clang.llvm.org
4. ClangStaticAnalyzer [Электронный ресурс] / https://clang-analyzer.llvm.org/
5. ClangTidy [Электронный ресурс] / https://clang.llvm.org/extra/clang-tidy/
6. Wikipedia : Profile-guided optimization [Электронный ресурс] / https://en.wikipedia.org/wiki/Profile-guided_optimization
7. Cozy, the collection synthesizer [Электронный ресурс] / https://cozy.uwplse.org/
8. Research Gate : Chameleon: Adaptive Selection of Collections [Электронный ресурс] / https://www.researchgate.net/publication/220751882_Chameleon_Adaptive_Selection_of_Collections