MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 29, NO. 3, 2022
journal homepage: www.mais-journal.ru

THEORY OF COMPUTING

Transformation of C Programming Language Memory Model into Object-Oriented Representation of EO Language

A. I. Legalov[1], Y. G. Bugayenko[2], N. K. Chuykin[1], M. V. Shipitsin[3], Y. I. Riabtsev[1], A. N. Kamenskiy[1]

DOI: 10.18255/1818-1015-2022-3-246-264

[1]National Research University Higher School of Economics, 20, Myasnitskaya str., Moscow 101000, Russia.
[2]Huawei Russian Research Institute, 7-9, Smolenskaya sq., Moscow 121099, Russia.
[3]ITMO University, 49 bldg. A, Kronverksky Pr., St. Petersburg 197101, Russia.

MSC2020: 68N15, 68Q10, 68N20
Research article
Full text in Russian

Received August 7, 2022
After revision September 1, 2022
Accepted September 2, 2022

The paper analyzes the possibilities of transforming C programming language constructs into objects of EO programming language. The key challenge of the method is the transpilation from a system programming language into a language of a higher level of abstraction, which doesn't allow direct manipulations with computer memory. Almost all application and domain-oriented programming languages disable such direct access to memory. Operations that need to be supported in this case include the use of dereferenced pointers, the imposition of data of different types in the same memory area, and different interpretation of the same data which is located in the same memory address space. A decision was made to create additional EO-objects that directly simulate the interaction with computer memory as in C language. These objects encapsulate unreliable data operations which use pointers. An abstract memory object was proposed for simulating the capabilities of C language to provide interaction with computer memory. The memory object is essentially an array of bytes. It is possible to write into memory and read from memory at a given index. The number of bytes read or written depends on which object is being used. The transformation of various C language constructs into EO code is considered at the level of the compilation unit. To study the variants and analyze the results a transpiler was developed that provides necessary transformations. It is implemented on the basis of Clang, which forms an abstract syntax tree. This tree is processed using LibTooling and LibASTMatchers libraries. As a result of compiling a C program, code in EO language is generated. The considered approach turns out to be appropriate for solving different problems. One of such problems is static code analysis. Such solutions make it possible to isolate low-level code fragments into separate program objects, focusing on their study and possible transformations into more reliable code.

Keywords: program transformation; procedural programming; object-oriented programming; transpilation; compilation; programming languages

INFORMATION ABOUT THE AUTHORS

Alexander I. Legalov correspondence author	orcid.org/0000-0002-5487-0699. E-mail: alegalov@hse.ru Doctor of Technical Sciences, Professor.
Yegor G. Bugayenko	orcid.org/0000-0001-6370-0678. E-mail: yegor256@huawei.com Director of system programming laboratory.
Nickolay K. Chuykin	orcid.org/0000-0001-9645-5525. E-mail: nchuykin@hse.ru PhD student.
Maksim V. Shipitsin	orcid.org/0000-0002-4957-5695. E-mail: mvshipitsin@edu.hse.ru Master's student.
Yaroslav I. Riabtsev	orcid.org/0000-0001-5530-3752. E-mail: yairyabtsev@edu.hse.ru Bachelor.
Andrey N. Kamenskiy	orcid.org/0000-0001-5197-310X. E-mail: ankamenskiy_1@edu.hse.ru Bachelor.

For citation: A. I. Legalov, Y. G. Bugayenko, N. K. Chuykin, M. V. Shipitsin, Y. I. Riabtsev, and A. N. Kamenskiy, "Transformation of C Programming Language Memory Model into Object-Oriented Representation of EO Language", *Modeling and analysis of information systems*, vol. 29, no. 3, pp. 246-264, 2022.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 29, № 3, 2022
сайт журнала: www.mais-journal.ru

THEORY OF COMPUTING

Трансформация модели памяти языка программирования C в объектно-ориентированное представление на языке EO

А. И. Легалов[1], Е. Г. Бугаенко[2], Н. К. Чуйкин[1], М. В. Шипицин[3], Я. И. Рябцев[1], А. Н. Каменский[1]

DOI: 10.18255/1818-1015-2022-3-246-264

[1]НИУ «Высшая школа экономики», ул. Мясницкая, д. 20, г. Москва, 101000 Россия.
[2]Российский исследовательский институт Huawei, Смоленская площадь, 7-9, г. Москва, 121099 Россия.
[3]Национальный исследовательский университет ИТМО , Кронверкский пр., д. 49, лит А., г. Санкт-Петербург, 197101 Россия.

УДК 004.4'4
Научная статья
Полный текст на русском языке

Получена 7 августа 2022 г.
После доработки 1 сентября 2022 г.
Принята к публикации 2 сентября 2022 г.

В работе проводится анализ возможностей трансформации конструкций языка программирования C в объекты языка программирования EO. Особенностью рассматриваемого подхода является транспиляция из языка системного программирования в язык с более высоким уровнем абстракции, не поддерживающий непосредственные манипуляции с компьютерной памятью. К действиям, которые в этом случае необходимо поддерживать, относится использование разыменованных указателей, наложение данных различного типа на одну и ту же область памяти, а также различная интерпретация одних и тех же данных, расположенных в одном и том же адресном пространстве памяти. Принято решение о создании дополнительных EO-объектов, напрямую имитирующих непосредственную работу с компьютерной памятью в языке C и инкапсулирующих ненадежные операции с данными посредством указателей. Для отображения возможностей языка C, обеспечивающих взаимодействие с компьютерной памятью, предлагается абстрактный объект-память. Он представляет собой массив байт, при работе с которым возможны запись и чтение по заданному индексу. Для исследования вариантов и анализа полученных результатов разработан транспилятор, обеспечивающий необходимые трансформации. Он реализован на основе Clang, который формирует абстрактное синтаксическое дерево. Обработка этого дерева осуществляется с использованием библиотек LibTooling и LibASTMatchers. Рассмотренный подход оказывается целесообразным при решении ряда задач, к одной из которых относится статический анализ кода. Подобные решения позволяют выделить в отдельные программные объекты низкоуровневые фрагменты кода, акцентировав внимание на их изучении и возможных преобразованиях в более надежный код.

Ключевые слова: трансформация программ; процедурное программирование; объектно-ориентированное программирование; транспиляция; компиляция; языки программирования

ИНФОРМАЦИЯ ОБ АВТОРАХ

Александр Иванович Легалов автор для корреспонденции	orcid.org/0000-0002-5487-0699. E-mail: alegalov@hse.ru доктор технических наук, профессор.
Егор Георгиевич Бугаенко	orcid.org/0000-0001-6370-0678. E-mail: yegor256@huawei.com директор лаборатории системного программирования.
Николай Константинович Чуйкин	orcid.org/0000-0001-9645-5525. E-mail: nchuykin@hse.ru аспирант.
Максим Владимирович Шипицин	orcid.org/0000-0002-4957-5695. E-mail: mvshipitsin@edu.hse.ru магистрант.
Ярослав Иванович Рябцев	orcid.org/0000-0001-5530-3752. E-mail: yairyabtsev@edu.hse.ru бакалавр.
Андрей Николаевич Каменский	orcid.org/0000-0001-5197-310X. E-mail: ankamenskiy_1@edu.hse.ru бакалавр.

Для цитирования: A. I. Legalov, Y. G. Bugayenko, N. K. Chuykin, M. V. Shipitsin, Y. I. Riabtsev, and A. N. Kamenskiy, "Transformation of C Programming Language Memory Model into Object-Oriented Representation of EO Language", *Modeling and analysis of information systems*, vol. 29, no. 3, pp. 246-264, 2022.

Введение

В настоящее время продолжаются исследования, направленные на создание новых методов и технологий разработки программного обеспечения (ПО), позволяющих с большей эффективностью поддерживать требуемые критерии качества. Зачастую эти методы расширяют или модифицируют существующие парадигмы, что ведет к созданию новых языков программирования и способов разработки программ. Проводимые исследования затрагивают популярные подходы, например, объектно-ориентированное программирование (ООП). В частности, в качестве альтернативного пути развития предлагается использование «элегантных объектов» [1—3]. Для апробации данного стиля предложен язык программирования EOLANG[1] (другое принятое в настоящее время название языка — EO) [4—6], одной из особенностей которого является попытка формализовать описание объектов с использованием φ-исчисления [4, 7]. Формализация, по мнению авторов, позволяет не только разрабатывать надежный код, но и более эффективно поддерживать статический анализ программ, написанных на данном языке.

Инструментальная поддержка статического анализа для языка программирования EO реализуется в рамках проекта Polystat [8]. Создаваемый анализатор предполагается использовать для анализа программ, написанных и на других языках программирования, при условии их трансформации в EO. В связи с этим решение проблемы трансформации из различных языков программирования в EO представляет определенный интерес для решения задач, связанных с повышением надежности программного обеспечения.

Вместе с тем формализация языковых средств ведет к повышению уровня абстракции, что не всегда позволяет однозначно отобразить конструкции, реализованные во многих универсальных языках программирования. Ряд конструкций ориентированы на прямое использование компьютерных ресурсов и являются ненадежными, если рассматривать их с позиций языков, ориентированных на прикладное объектно-ориентированное программирование. Наличие ненадежных конструкций затрудняет поиск ошибок во время статического анализа. С другой стороны, многие решения, принимаемые программистом, можно реализовать, не используя низкоуровневые средства, то есть, повысить уровень абстракции при сохранении исходной семантики. Это позволяет проводить последующий статический анализ для более формализованного или инкапсулированного представления. При этом основной проблемой является отображение ненадежных программных конструкций менее формализуемого языка в объекты, используемые в более формализуемом языке.

Одним из языков, для которого повышение уровня абстракции при статическом анализе может оказаться интересным, является язык программирования C. В целом, его конструкции достаточно просты. Однако их использование зачастую сильно ориентировано на архитектуру уровня системы команд, что не допускает непосредственную трансформацию в объекты языка EO. Во многом это обусловлено тем, что в C имеется непосредственное отображение данных на линейную память компьютера. Это часто не позволяет отделить средства языка от памяти компьютера и рассматривать их как независимые сущности. Большая часть проблем при этом выливается в использование указателей, манипуляции которыми зачастую не связаны с непосредственной обработкой данных и могут интерпретироваться как некоторые действия с областями компьютерной памяти. Многие из этих манипуляций обладают поведением, описанным в документации по языку как непредсказуемое. Поэтому при трансформации из языка C нужно учитывать наличие в нем линейной памяти и определенным образом отображать на нее объекты EO. То есть, наряду с формированием объектов EO, необходимо иметь возможность моделировать поведение памяти языка C как набора байт, а также восстанавливать из байт памяти объекты, над которыми можно осуществлять обработку данных в EO. Ряд подходов, позволяющих использовать в EO не объектно-ориентированные конструкции, были предложены в работе [9].

[1] `https://www.eolang.org`

Несмотря на видимую простоту языка C и имеющуюся аналогию с методами трансформации, используемыми в различных компиляторах, решение задачи, связанной с трансформацией и последующим статическим анализом, требует дополнительных исследований, что обуславливается спецификой семантической модели языка программирования EO, представляющей объекты на более высоком уровне абстракции. Наряду с задачей отображения не объектно-ориентированных конструкций возникают вопросы, связанные с возможностью полной или частичной замены конструкций языка C, ориентированных на использование линейной памяти, объектами, которые бы напрямую отображали семантически значимые понятия исходной программы без необходимости рассматривать более низкий уровень их отображения. Это позволило бы повысить уровень абстракции программы перед проведением ее статического анализа и получить на выходе программу, которая в большей степени была бы ориентирована на объектно-ориентированное представление предметной области решаемой задачи.

В работе проводится анализ вариантов трансформации программных конструкций языка программирования C в объекты языка программирования EO. Рассматриваются проблемы, связанные с непосредственным отображением на уровне единицы компиляции. Для исследования вариантов и анализа полученных результатов реализован транспилятор, обеспечивающий необходимые трансформации.

1. Специфика языка программирования EO

Язык программирования EO относится к бестиповым объектно-ориентированным языкам. Основной его сущностью является объект, который вместо методов содержит другие объекты (подобъекты). Объект или подобъект может получать дополнительные атрибуты, которые также являются объектами. Эти атрибуты по сути эквивалентны объектам, подключаемым через параметры в конструкторах традиционных языков объектно-ориентированного программирования. При этом через них обеспечивается передача любых объектов без дополнительного анализа их типа, что и позволяет говорить о бестиповом характере языка. Отсутствие идентификации типов у передаваемых атрибутов является спецификой EO. Поддержка однозначности процессов обработки данных обеспечивается через идентификацию внешних объектов, каждый из которых имеет уникальное имя. Во многом подобный механизм однозначности совпадает с уникальностью имен операций в бестиповых языках, например, в ассемблерах.

Любой объект может содержать подобъекты. Дополнительно уникальная идентификация поддерживается различием имен на каждом из уровней вложения подобъектов. Допускается произвольная вложенность объектов. Идентификация подобъектов реализуется через использование составных имен или их именование относительно текущего расположения. Это позволяет при передаче разыменованных атрибутов различных объектов обращаться к именам подобъектов, что по сути поддерживает «утиную» типизацию, определяющую полиморфизм. В отличие от традиционных бестиповых императивных систем, в которых любое обращение выполняемой операции к памяти осуществляется без анализа контекста данных, обращение через разыменованный атрибут к объекту, который не содержит целевой подобъект, ведет к прерыванию программы во время выполнения. Во время компиляции подобные ситуации напрямую не идентифицируются.

Операция над объектом, называемая «датаизацией», обеспечивает возврат результата в виде некоторого объекта, который может являться результатом множественного обращения как к внутренним подобъектам, так и к объектам, передаваемым через атрибуты, включая их подобъекты. По сути это обеспечивает реализацию любых алгоритмов, описывая их в рамках специфической для EO интерпретации.

Основу языка составляют атомарные объекты, являющиеся аналогами базовых типов данных и операций над ними. При этом сами операции реализуются как подобъекты. В качестве примеров можно привести целочисленный объект `int` и объект с плавающей точкой `float`. В каждом

из них имеется подобъект `plus`, реализующий операцию сложения в соответствии с семантикой родительского объекта. Формирование новых объектов может осуществляться с использованием: композиции из уже существующих объектов; созданием новых объектов клонированием существующих объектов; применением декорирования, когда новый объект напрямую подключает поля уже существующего объекта.

Язык EO ориентирован на ленивые вычисления, что обуславливается разделением фаз инициализации объектов и обращения к ним во время выполнения программы. Исходя из этого допускается частичная инициализация объектов атрибутами. Во многом выполнение программы ассоциируется с функциональным программированием, где вместо функций осуществляется взаимодействие связанных друг с другом объектов. Однако для расширения возможностей поддержки императивных языков, необходимой для повышения гибкости статического анализатора, разработан ряд «серых» объектов, инкапсулирующих ненадежные конструкции. Эти объекты поддерживают последовательность действий, организацию циклов, упрощенные безусловные переходы, изменчивость внутреннего состояния. Это позволяет использовать EO в качестве выходного представления при компиляции с различных языков программирования, поддерживающих императивный стиль.

2. Отображение модели памяти языка C в объекты EO

Особенностью рассматриваемого подхода является транспиляция из языка системного программирования в язык с более высоким уровнем абстракции, не поддерживающий непосредственные манипуляции с компьютерной памятью. Отсутствие подобной поддержки свойственно практически всем языкам прикладного и предметно-ориентированного программирования. Вместе с тем, существует достаточно примеров использования высокоуровневых языков для решения задач, ориентированных на низкоуровневое программирование. В основном они связаны с эмуляцией низкоуровневых конструкций высокоуровневыми средствами.

В частности, можно отметить ряд транспиляторов, осуществляющих преобразования с одних языков высокого уровня в другие языки, поддерживающих выполнение программ в режиме интерпретации. Зачастую подобные решения используются как web-приложения, функционируя в среде Интернет браузера. В качестве примера можно привести компилятор с языка программирования Оберон-7 в JavaScript [10], который используется для изучения программирования в интерактивном режиме [11].

Достаточно часто высокоуровневые языки программирования используются для разработки эмуляторов компьютерных архитектур, запускаемых в браузерах. Несмотря на невысокую производительность, они эффективно используются как для запуска старых игр, так и в учебном процессе. Примерами таких систем являются эмуляторы ZX Spectrum [12] и Nintendo NES [13], написанные на JavaScript. В качестве еще одного нетипичного примера можно привести эмуляцию на JavaScript операционной системы [14]. В этой ситуации также часто используются реализации, поддерживающие Интернет-технологии.

2.1. Подходы к транспиляции из C в EO

В рассматриваемой ситуации транспиляция из языка программирования C в язык EO связана с использованием полученного представления для статического анализа. Существуют различные варианты применения статических анализаторов, среди которых можно отметить:

- определение некорректного стиля программирования, связанного с особенностями использования языковых конструкций конкретного языка программирования;
- выявление функциональных ошибок, обусловленных некорректными алгоритмическими решениями;
- поиск ошибок уязвимости, приводящих к некорректной работе с данными, например, выходу за границы массивов;

- анализ ситуаций, связанных с некорректным использованием ресурсов памяти и другие.

Ряд методов анализа, например, некорректный стиль кодирования, может быть связан с конкретным языком программирования. Однако анализ многих ошибок не привязан к нему, что позволяет в качестве решения использовать универсальный статический анализатор, независимый от языка программирования.

Непосредственное использование только атомарных объектов, поддерживающих безопасное программирование, возможно и при создании транспилятора из C в EO. Однако в этом случае реализация ряда конструкций языка C может получиться громоздкой, что затруднит статический анализ. Вместо этого принято решение о создании дополнительных «серых» EO-объектов, напрямую имитирующих непосредственную работу с компьютерной памятью в языке C. К действиям, которые в этом случае необходимо поддерживать, относятся использование и разыменование указателей, наложение данных различного типа на одну и ту же область памяти, а также различная интерпретация одних и тех же данных, расположенных в одном и том же адресном пространстве памяти. Разработка специальных «серых» объектов для последующего статического анализа подобных ситуаций позволяет отделить задачи, связанные с ненадежным программированием, от задач общего вида, присущих различным языкам программирования. Это позволяет отдельно решать задачу трансформации ненадежных объектов в безопасные и обратно. Для отображения компьютерной памяти в языке реализован специальный объект `ram`, на основе которого и реализована трансформация ряда конструкций языка C.

2.2. Отображение атомарных объектов

Моделирование компьютерной памяти, связанное с отображением на нее EO-объектов, не отменяет манипуляции непосредственно с атомарными объектами. Поэтому необходим механизм, позволяющий отображать EO-объекты на компьютерную память и восстанавливать их перед выполнением операций. При этом нужно учитывать различие между атомарными конструкциями языка C и атомарными объектами языка EO.

К атомарным конструкциям языка программирования C относятся базовые типы данных, переменные, сформированные на их основе, а также операции с аргументами базового типа. Учитывая то, что в EO отсутствуют абстракции типов в прямом понимании, преобразование базовых типов языка C не происходит. Вместо этого осуществляется формирование объектов непосредственно для переменных. Тип переменной при этом совпадает с соответствующим абстрактным атомарным объектом, на основе которого осуществляется построение соответствующего конкретного объекта. Для каждого базового типа языка C имеется аналог в виде соответствующего объекта EO. Операциям языка C над аргументами базового типа соответствуют подобъекты EO, расположенные в соответствующих атомарных объектах. При отсутствии таких аналогов они формируются на основе объектов-оберток над одним или несколькими атомарными объектами (то есть путем моделирования новых объектов, покрывающих недостающие конструкции языка C).

2.2.1. Отображение переменных базовых типов

Существующим в языке C базовым типам, (`int`, `long int`, `double` и т.д.) в EO имеются аналоги в виде соответствующих абстрактных объектов (`int`, `long`, `float` и др.). Каждый из таких объектов EO содержит множество подобъектов, которые по функциональному назначению соответствуют методам традиционных языков объектно-ориентированного программирования. Для более удобного отображения над атомарными объектами EO сформирована дополнительная обертка, которая позволяет их рассматривать как обычные переменные. Подобъекты-методы при этом также обернуты во внешние объекты, обеспечивая тем самым полное восприятие их в качестве операций над базовыми типами.

Исходя из этого объявление переменной любого базового типа языка C отображается в создание нового объекта на основе соответствующего атомарного объекта. Например, описание переменной «`int x;`» трансформируется в порождение следующего объекта EO:

```
int-32 > x
```

где `int-32` - обертка над абстрактным EO-объектом `int`, имитирующая целочисленный тип языка C. Аналогичным образом осуществляется прямое отображение в атомарные объекты EO всех других базовых типов и созданных на их основе скалярных переменных языка C.

2.2.2. Использование объекта памяти для базовых типов

Для отображения возможностей языка C, обеспечивающих взаимодействие с компьютерной памятью, используется «серый» абстрактный объект `ram`. Он представляет собой массив байт, при работе с которым возможны запись и чтение по заданному индексу указанного количества байт. Количество байт определяется в зависимости от того, из какого объекта происходит запись или при формировании какого объекта осуществляется чтение. При записи в память данных из некоторого объекта происходит их разыменование. Поэтому при чтении необходимо явно указать, какой тип объекта восстанавливается. Операции чтения-записи реализованы для работы со всеми атомарными объектами EO.

Наличие статической типизации в языке C позволяет знать тип объекта, для которого происходит запись в память с разыменованием, и явно указывать это при обращении к соответствующему объекту EO. Разыменование поддерживается соответствующими объектами-оболочками. Например, запись в объект памяти `ram` отображения целочисленной переменной `int` как целочисленного объекта `int-32` с именем `obj` по некоторому адресу `addr` описывается следующим образом:

```
write-as-int32
  addr
  obj
```

где EO-объект `write-as-32` записывает в память по адресу `addr` разыменованное содержимое целочисленного объекта `obj` как 32-разрядное битовое поле. Следует отметить, что адрес `addr` непосредственно связан с заданным объектом памяти, что будет описано ниже.

Аналогичная операция восстановления разыменованного значения в целочисленный объект `obj2` будет представлена следующим образом:

```
read-as-int32 > obj2
  addr
```

2.2.3. Отображение операций над базовыми типами

В языке EO обработка атомарных объектов осуществляется путем обращения к подобъектам-методам, расположенным внутри этих атомарных объектов. Для имитации операций с дополнительными аргументами осуществляется передача этим подобъектам объектов-атрибутов. Для удобства отображения данные конструкции при трансформации из C в EO были обернуты во внешние объекты, что обеспечило их восприятие, аналогичное функциональной нотации. К особенностям языка EO также следует отнести возможность представления программы в скобочной и ступенчатой форме, которые могут использоваться как совместно, так и раздельно. Для отображения конструкций языка C в основном использовалась ступенчатая нотация. В частности, операция сложения языка C «`x + y`» отображается в EO следующим образом:

```
plus
  x
  y
```

где тип объектов x и y восстановлен после выборки из объекта-памяти ram.

Использование объекта-памяти ведет к дополнительным промежуточным манипуляциям с данными при вычислении выражений, связанным с чтением-записью. Например, при трансформации оператора c = a + b, где операнды имеют тип int, сформируется следующий код на EO:

```
write-as-int32
  g-c
  plus
    read-as-int32
      g-a
    read-as-int32
      g-b
```

где g-a, g-b, g-c — адреса областей в памяти, отведенной для хранения переменных a, b и c. То есть, в начале при чтении из областей памяти с адресами g-a и g-b восстанавливаются объекты типа int-32. После этого осуществляется их сложение объектом-оболочкой plus. В результате сложения формируется новый объект типа int-32, значение которого в виде набора байт записывается по адресу g-c.

2.3. Указатели и операции с их использованием

Указатели широко используются в языке C. Они являются средством непосредственного обращения к компьютерной памяти, что, с одной стороны, ведет к эффективной организации вычислений, а с другой — к ненадежному программированию. В большинстве современных языков, особенно тех, которые ориентированы на прикладное программирование, непосредственное использование указателей заменено на более надежные и безопасные конструкции. Это также касается и языка EO, в котором все изначальные манипуляции данными и их обработка осуществляются только на уровне объектов.

Имитация указателей осуществляется за счет их представления в виде атомарных целочисленных объектов, с каждым из которых ассоциируется индекс в линейной памяти объекта ram. Значение указателя воспринимается как индекс к линейной области объекта-памяти, что позволяет косвенно обратиться к любому разыменованному объекту. Сами указатели уже не являются типизированными. Они просто указывают на местоположение разыменованных объектов любого типа. Восстановление типа объектов осуществляется при их чтении в соответствии с информацией, получаемой в ходе транспиляции исходной программы на языке C.

Указатель может принимать любое положительное целочисленное значение. Он занимает 8 байт при хранении в объекте-памяти, восстанавливаясь при чтении из нее в атомарный объект int-64. Как и в языке C, указатель может иметь пустое значение, соответствующее нулевому значению (NULL) языка C. Учитывая его частое использование в различных операциях, также трактуемое при преобразованиях как ложное или нулевое значение, имеет смысл оставить нулевое значение и в EO. Это ведет к тому, что данные в объекте-памяти нужно размещать не с нулевого адреса, а, например, с адреса, равного 8 или больше. Тогда проверка на нулевой указатель будет осуществляться аналогично проверке в языке C.

Операции над указателями языка C трансформируются в манипуляции с целочисленными объектами языка EO. Учитывая их разыменование, при выполнении этих операций необходимо

учитывать размер типа, доступный во время транспиляции программы на языке C. При смещении указателя на единицу в языке C, его значение в EO (как и в адресном пространстве памяти C) необходимо изменять на размер типа. Например, пусть имеется описание указателя:

```
int *ptr;
```

и его последующее использование в некотором контексте:

```
++ptr;
```

Тогда, учитывая размерность типа `int`, равную 4 байтам, значение объекта в EO, определяющее указатель, увеличится на 4. То есть, будет порожден следующий код.

```
write-as-int64
  ptr
  plus
    read-as-int64
      ptr
    4
```

3. Трансформация единицы компиляции

Единица компиляции языка C, предварительно обработанная препроцессором, используется для трансформации в EO. Именно на ее основе формируется файл, образующий пакет EO, который служит базовой компонентой для последующих манипуляций, включая статический анализ. Исходя из этого осуществляется однозначное отображение всех тех конструкций, которые могут содержаться в единице компиляции. К ним следует отнести:

- описания внешних глобальных и статических переменных;
- описания функций;
- объявления (прототипы) функций;
- объявления внешних переменных;
- описания составных типов данных;
- определения алиасов типов (typedef).

Трансформация каждой из этих конструкций требует специфических решений.

3.1. Отображение единицы компиляции языка C в объект EO

Отображение единицы компиляции языка C ведет к порождению EO-объекта с именем `global`, в котором размещаются EO-объекты, формируемые в ходе транспиляции. Помимо этого, сформированный объект содержит инициализацию подобъектов, отображающих глобальные и статические переменные языка C. При наличии в единице компиляции функции `main` также формируется запуск объекта `main` отображающего эту функцию. Для отображения внешних зависимостей в формируемом файле порождаются алиасы, имитирующие объявления внешних функций и переменных, а также объектов языка EO, разработанных для имитации ряда программных конструкций языка C и используемых в создаваемом пакете.

Общую схему EO-пакета, формируемого в ходе трансформации, можно представить следующим образом:

```
Задание имени формируемого EO-пакета
Алиасы подключаемых внешних объектов различного назначения
[args...] > global
```

```
Описания объектов, отображающих имитируемые конструкции языка C
seq > @
  Инициализация объектов, имитирующих глобальные и статические данные
  Запуск объекта main при наличии функции main
```

3.2. Отображение внешних переменных

К внешним относятся глобальные и статические переменные языка C. Следует отметить, что их прямое отображение в объекты EO не позволяет воспроизвести присущую языку C функциональность так как переменные в C непосредственно проецируются на компьютерную память. Это отображение может быть реализовано в C-программе с наложением областей памяти друг на друга и рядом других эффектов. Поэтому для имитации различных возможностей используется «серый» объект-память. Каждая внешняя переменная отображает свои данные на объект-память, размещаясь в выделенном адресном пространстве, которое определяется в процессе трансформации. Для хранения образов этих переменных на основе объекта `ram` формируется объект `global-ram`, который имитирует распределение и использование общей памяти данных. Глобальные и статические переменные размещаются в нем вперемешку по ходу транспиляции. Также на этом этапе в данной области распределяется адресное пространство под статические переменные, располагаемые внутри функций, расположенных в единице компиляции. После их размещения свободное адресное пространство `global-ram` используется для локальных данных функций. Она автоматически выделяется и освобождается при вызове каждой из функций. Начало области локальных данных фиксируется в специальном объекте, который инициализируется после определения адресного пространства, занимаемого глобальными и статическими данными.

3.3. Отображение вспомогательных объектов

Помимо этого, в объекте `global` располагаются дополнительные подобъекты, обеспечивающие поддержку различных действий:

1. Хранилище возвращаемого параметра (`return-ram`), принимающее значение результата, возвращаемой из функций. Создается на основе объекта (`ram`). Размер возвращаемого значения определяется во время компиляции и может задаваться в виде константы (например, при копировании возвращаемого значения в другие области памяти).
2. Фиксатор начала области локальных данных (`start-local`), который не изменяется при формировании этой области и используется для контроля выхода за ее нижнюю границу.
3. Фиксатор свободного локального подпространства, определяющий начало размещения локальных данных для очередной вызываемой функции (`empty-local`). Для каждой вызываемой функции в начале этого подпространства формируется и заполняется вызывающей функцией список фактических параметров, за которым следует список фиксируемых локальных данных функции.
4. Фиксатор размера списка фактических параметров (`param-size`), задающий объем памяти, выделяемой под передаваемые в вызываемую функцию фактические параметры.

Передаваемые фактические параметры располагаются в начале локальной области. После них следует область локальных переменных вызываемой функции. Размер этой области и ее начало определяются уже внутри данной функции. Начало области локальных параметров функции определяется суммированием значений `empty-local` и `param-size`, которые передаются в функцию в качестве атрибутов. В ходе выполнения функции ее список локальных данных может расширяться автоматически формируемыми массивами и временными промежуточными переменными.

3.4. Описания функций

Описания функций отображаются в соответствующие им объекты-функции. Они также размещаются в глобальном объекте. Каждый такой объект хранит необходимую информацию для доступа к областям глобальной памяти (`global-ram`). Для повышения эффективности доступа используется специальный объект, облегчающий манипуляцию общей памятью и содержащий вспомогательные артефакты. Для доступа к фактическим параметрам имеется фиксатор начала параметров (`param-start`), который устанавливается в значение текущего начала локальной области (`empty-local`), передаваемое в качестве атрибута. Другим атрибутом, передаваемым в функцию, является размер области фактических параметров (`param-size`). Его добавление к началу локальной области определяет базу для вычисления локальных переменных (`local-start`).

Значение, возвращаемое из функции, передается в хранилище возвращаемого параметра `return-ram`, а вызывающая функция переносит его в требуемую переменную. Константа `ret-size`, фиксирующая размер возвращаемого параметра, позволяет осуществить этот перенос между различными областями памяти, используя для этого байтовые копирующие операции.

4. Особенности отображения переменных языка C

В рамках целевой задачи, ориентированной на статический анализ программы, создаваемой в ходе транспиляции, нет особой необходимости полностью имитировать подходы, используемые при компиляции в низкоуровневое машинное представление. Однако ряд возможностей, допускаемых исходным языком, имеет смысл поддержать. Поэтому объект `global-ram` используется для хранения внешних переменных различного типа, а также имитирует возможности стека для отображения локальных переменных.

Размещение переменных связано с распределением адресного пространства. Оно во многом аналогично распределению памяти различными генераторами машинного кода. Для размещения переменной необходимо знать размер типа, который берется из абстрактного синтаксического дерева (AST), построенного с использованием clang, а также адрес свободной области в объекте-памяти `global-ram`. В большинстве случаев для различных типов данных особых проблем при распределении адресного пространства не возникает. Особенностью является то, что при размещении данных происходит разыменование типов. Поэтому при выборке данных и выполнении над ними различных операций необходимо обратное восстановление типов, которое осуществляется созданием атомарных объектов языка EO. Создаваемые объекты содержат подобъекты, осуществляющие вычисления, аналогичные методам традиционных языков ООП и эквивалентные операциям над данными языка программирования C.

4.1. Трансформация переменных базовых типов

При трансформации переменных базовых типов происходит распределение их в адресном пространстве соответствующих объектов байтовой памяти. Для идентификации размещаемых данных и фиксации их местоположения используется специальный объект `address`:

```
address > name
  addr
```

где `name` — имя переменной с учетом префикса, под которую выделяется память, `addr` — индекс в `global-ram`, определяющий начальный адрес для размещения байтового образа объекта.

Имена объектов формируются с добавлением префиксов и суффиксов в зависимости от контекста переменных в языке C. Это связано с тем, что в EO используется другой подход к формированию идентификаторов, а также с тем, что присутствует другая трактовка областей видимости, не столь сильно увязанная с блочной структурой, как в языке C. Имена глобальных объектов языка C, доступные в разных единицах компиляции, начинаются с префикса «`g-`», для статических объектов

используется префикс «s-», для объявлений внешних переменных — «e-», для локальных (в функциях) — «l-», для аргументов функции — «a-». Имена функций имеют префикс «f-».

Блочная структура языка C определяет наличие областей видимости имен. Поэтому возможно появление нескольких объектов EO с одинаковыми именами. Для их однозначной трактовки используется дополнительная идентификация в виде уникального суффикса. В качестве такого суффикса выступает номер объекта в AST, формируемом clang.

4.2. Трансформация структурных переменных

Структурные переменные в языке C создаются на основе описания соответствующих структурных типов. Основной спецификой полей в структуре является их смещение относительно ее начала. При отображении структуры на память достаточно знать адрес размещения и размер. Учитывая, что структурный тип имеет фиксированный размер, определяемый во время компиляции, выделение памяти для него проблем не вызывает.

Спецификой структурных переменных является наличие внутренних полей. Для задания значения смещений в порождаемом на EO коде формируются соответствующие константы, значения которых прибавляются к адресу размещения структуры в памяти, формируя тем самым окончательный адрес поля. Имена этих констант начинаются с префикса «st-». Далее следует имя структуры и, отделяемое от него знаком «-», имя поля. При наличии в структуре вложенных подструктур следует набор имен полей подструктур, разделяемых знаком «-».

Предположим, что имеется внешняя структурная переменная на языке C, описывающая прямоугольник:

```
// прямоугольник
struct rectangle {
  int x, y; // ширина, высота
};
struct rectangle r;
```

Тогда в ходе трансформации будет получен следующий код на EO, задающий имена полей и их смещения:

```
0 > st-rectangle-x
4 > st-rectangle-y
```

Целочисленные константы, в зависимости от типов полей, задают смещения в байтах от начала структуры. Имя каждого из полей начинается с префикса структуры, за которым следует имя структуры, а далее идет имя поля. Все имена отделяются друг от друга знаком «-».

При создании структурной переменной происходит выделение памяти, начиная с первого свободного адреса. Начальный адрес переменной фиксируется в ее описании. В данном случае структурная переменная r размещается, начиная со свободного адреса, например, равного восьми:

```
address > g-r
  global-ram
  8
```

Префикс «g-» указывает на ее размещение в области глобальных переменных.

Обращение к полю переменной в коде программы, например, к полю r.x, осуществляется с добавлением требуемого смещения к адресу, определяющему начальное местоположение структурной переменной в памяти:

```
plus
  g-r
  st-rectangle-x
```

4.3. Трансформация объединений и структур, содержащих объединения

Основным отличием объединения является то, что его размер определяется вложенным программным объектом, имеющим максимальный размер. Поэтому выделение памяти аналогично тому, как она выделяется для структуры. Особенность лишь в том, что все альтернативные поля начинаются с одного и того же адреса, что необходимо учитывать при формировании описания локальных полей. Рассмотрим трансформацию структуры, описывающую геометрическую фигуру, состоящую из прямоугольника и треугольника:

```
// прямоугольник
struct rectangle {
     int x, y; // ширина, высота
};
// треугольник
struct triangle {
    int a, b, c; // стороны
};
// фигура
struct figure {
    int key;
    union {
        struct rectangle r;
        struct triangle t;
    };
};
```

В ходе трансформации будут получены следующие описания смещений для объединения `fig`:

```
0 > st-rectangle-x
4 > st-rectangle-y
0 > st-triangle-a
4 > st-triangle-b
8 > st-triangle-c
0 > un-94438030510136-r
0 > un-94438030510136-t
4 > st-figure-field747
0 > st-figure-key
```

Числовой идентификатор, сформированный на основе уникального номера узла в `clang` подставляется вместо имени для неименованного объединения или структуры. Все поля внутри любого объединения имеют нулевое смещение. Внешнее имя неименованной структуры или объединения начинается с обозначения `field`, за которым следует уникальный числовой суффикс (в примере это 747), автоматически порождаемый `clang` для неименованных полей. Обращение к полю объединения, например к `f.r`, осуществляется аналогично обращению к полю структуры, но с учетом двойной вложенности. То есть адрес данного поля будет сформирован следующим образом:

```
plus
  plus
    l-f
    st-figure-field747
  un-94438030510136-r
```

4.4. Трансформация переменных перечислимого типа

Переменные перечислимого типа отображаются в целочисленные значения. Эти значения известны в процессе компиляции и доступны из сформированного AST. Следовательно, для их размещения в памяти достаточно выделить размер, используемый для хранения целого числа. Сами перечислимые значения можно напрямую представлять соответствующими числовыми значениями без приведения их к именованным значениям. То есть поступать с ними так же, как с константами и константными выражениями.

4.5. Трансформация массивов

Для массивов фиксированной размерности, размещаемых в статической памяти (внешней и внутренней), трансформация происходит аналогично выделению памяти для переменных базовых типов. Это обуславливается знанием их размера во время компиляции. Поэтому зарезервировать заданное число байт достаточно просто. Для идентификации также используется объект `address`, задающий адрес первого элемента массива. Подобное решение используется для массивов любой размерности. Отличие многомерных массивов проявляется во время их обработки и связано с реализацией формул пересчета индексов в адрес.

4.6. Инициализации глобальных и статических переменных

Традиционно глобальные и статические переменные инициализируются нулями или начальными значениями до запуска функции `main`. Поэтому при трансформации в EO инициализация осуществляется в декораторе объекта `global`, который охватывает глобальное подпространство и подпространство функций транспилируемой единицы компиляции. При наличии в единице компиляции функции `main` она также запускается в конце последовательности инициализаций.

5. Трансформация функций

Все функции языка C, встречающиеся в единице компиляции, преобразуются в объекты-функции. В качестве атрибутов каждой функции передаются адрес глобальной памяти, с которого начинается список формальных параметров, заполненный вызывающей функцией, и размер памяти, занимаемой ее списком фактических параметров. Память под локальные параметры функции выделяется после области ее фактических параметров. Подобная унификация упрощает генерацию кода, сохраняя семантическую преемственность. В целом, процесс аналогичен трансформации в архитектуры уровня системы команд. Учитывая, что EO является бестиповым языком, такой подход вполне допустим.

При наличии в функции возвращаемого параметра его значение, перед завершением выполнения объекта-функции, копируется в буфер результата выполнения функции `return-ram`.

Формирование адресов (индексов) параметров и локальных переменных может осуществляться аналогично тому, как это делается для статических переменных. То есть путем задания имени и адреса.

При трансформации функций с фиксированным числом формальных параметров размер передаваемых параметров известен заранее. Поэтому при генерации кода объем данных, копируемых из буфера аргументов в локальную память параметров задаёт сама вызываемая функция. Если же функция задана с переменным числом параметров, то в этом случае объем получаемых данных определяется после заполнения вызывающей функцией буфера аргументов и установления

текущего размера буфера в соответствующем объекте. Тогда вызываемая функция обращается к объекту, хранящему размер данных в буфере аргументов и копирует данные в локальную память аргументов, в соответствии с этим значением.

5.1. Трансформация выражений над указателями

В языке C имеются разнообразные операции, выполняемые над указателями. При трансформации в EO осуществляется представление указателей в виде целочисленного объекта с потерей типа. Сам тип при выполнении манипуляций с указателями не требуется. Поэтому большинство операций выполняется аналогично тому, как выполняются операции с целыми числами, но с возможным учетом размеров типов данных, над которыми выполняются эти манипуляции (то есть с умножением на число байт, отводимых под тот или иной тип). Непосредственно приведение к нужному типу данных осуществляется только перед выборкой значения из глобальной памяти и выполнением над ним требуемых манипуляций или преобразований, как это обычно и происходит при бестиповом подходе к вычислениям.

5.2. Трансформация операторов

В языке EO имеется набор объектов, которые реализуют функции, аналогичные операторам языка C. Поэтому трансформация операторов в принципе не вызывает проблем. В частности, есть прямое отображение условного оператора, оператора цикла `while-do`, составного оператора (блока) и ряда других. В тех случаях, когда прямая трансформация невозможна, осуществляется отображение через совокупность существующих EO-объектов. В частности, прочие операторы цикла моделируются через объекты, задающие условия и объект `goto`, особенностью которого является не выполнение произвольных переходов, а имитация операторов `continue` и `break`. Для реализации оператора `switch` используются объекты `if`.

Когда трансформация конструкций языка C невозможна (например, при наличии в коде операторов безусловного перехода), формируются псевдообъекты, не имеющие реализации в EO. Предполагается, что окончательное решение по их трансформации или преобразованию в другой код будет приниматься при статическом анализе.

6. Пример трансформации единицы компиляции

Рассмотрим основные особенности кода на EO, порождаемого при транспиляции простой программы, написанной на языке программирования C:

```
int a = 5, b;
int main() {
  int c = 10;
  b = a + b;
  return 0;
}
```

Порождаемый EO-код разбит на фрагменты и прокомментирован. Отступы в два пробела и образованные при этом «ступеньки» в каждом из фрагментов кода соблюдены в соответствии с синтаксисом языка EO и используемой ступенчатой записью.

Первоначально к любой создаваемой программе подключаются внешние атомарные объекты и объекты-обертки, используемые в порождаемом коде. Также формируется имя создаваемого пакета.

```
+alias c2eo.coperators.address
+alias org.eolang.goto
+alias c2eo.coperators.plus
+alias c2eo.coperators.ram
+alias c2eo.coperators.read-as-int32
+alias c2eo.coperators.write-as-int32
+package c2eo.src.example.ex
```

Далее создается объект `global`, имитирующий единицу компиляции. Данный объект служит точкой запуска программы. Поэтому через него передаются аргументы командной строки в функцию `main` при их наличии.

```
[args...] > global
```

Для имитации внешней памяти, используемой для хранения глобальных и статических переменных, на основе объекта `ram`, импортируемого через алиасы, создается объект `global-ram`, имитирующий глобальную память языка C. Его размер в текущей реализации равен 8192 байта. Помимо этого, создается еще один общий объект-память аналогичного размера `return-ram`, который используется для возврата результата из любой функции, возвращающей параметр. Также порождаются объекты, используемые для фиксации свободной позиции в глобальной памяти `empty-global-position`, записи размера параметра, возвращаемого из функции (`return-mem_size`), и начальный адрес возвращаемого параметра `return`, всегда равный нулю.

```
ram > global-ram
  8192
memory > empty-global-position
ram > return-ram
  8192
memory > return-mem_size
address > return
  return-ram
  0
```

После создания объектов, моделирующих глобальную память и память возврата данных из функций, осуществляется распределение адресного пространства под глобальные и статические переменные, начиная с адреса, равного 8, и оставляя резервным нулевой адрес для фиксации пустого указателя. Для целочисленных переменных отводится по 4 байта.

```
address > g-a
  global-ram
  8
address > g-b
  global-ram
  12
```

Далее следуют описания функций, каждая из которых имеет по два входных объекта-атрибута. Атрибут `param-start` передает в функцию адрес, начиная с которого размещаются фактические параметры. Атрибут `param-size` передает значение, задающее размер области памяти, занимаемой фактическими параметрами. Это позволяет вычислить адрес `local-start`, с которого начинается

размещение локальных данных функции. Следует отметить, что вместо стека локальные данные размещаются по возрастанию вслед за данными, уже размещенными в глобальной памяти. То есть стек функций растет вверх.

```
[param-start param-size] > main
  plus > local-start
    param-start
    param-size
  plus > empty-local-position
    local-start
    4
  address > l-c
    global-ram
    plus
      local-start
      0
```

После распределения памяти под локальные данные запускается тело функции. Оно начинается с объекта `goto`, который осуществляет переход на метку, передаваемую объекту в качестве параметра. Его использование в теле функции позволяет сформировать множественные возвраты из нее. В примере имеется только объект, имитирующий функцию `main` языка C.

```
goto > @
  [goto-return-label]
    seq > @
      write-as-int32
        l-c
        10
      write-as-int32
        g-b
        plus
          read-as-int32
            g-a
          read-as-int32
            g-b
      write-as-int32
        return
        0
      goto-return-label.forward TRUE
      TRUE
```

После описания объектов-функций осуществляется инициализация объектов, имитирующих глобальные и статические переменные. При наличии в единице компиляции на языке C функции `main` осуществляется запуск соответствующего объекта EO.

```
seq > @
  write-as-int32
    g-a
    5
  write-as-int32
    g-b
    0
  main
    16
    0
  TRUE
```

7. Инструментальная поддержка транспиляции

Для апробации рассмотренных решений в рамках проекта Polystat ведется разработка транспилятора из языка программирования C в язык EO [15]. Он реализован на основе системы инструментальной поддержки Clang [16], которая обеспечивает формирование абстрактного синтаксического дерева. Обработка AST осуществляется с использованием библиотек LibTooling и LibASTMatchers [17]. В результате компиляции программы на языке C на выходе формируется код на языке EO, который и используется для статического анализа.

Следует отметить, что до запуска транспиляции из C в EO осуществляется запуск препроцессора языка C, который преобразует используемые директивы. Это позволяет перед трансформацией в EO получить уже обработанный исходный текст и не обрабатывать на последующем этапе директивы препроцессора. Как и Polystat, проект транспилятора реализован под открытой лицензией MIT.

Заключение

Разработка методов трансформации низкоуровневых языковых систем связана с решением множества задач моделирования ненадежных языковых средств с использованием высокоуровневых конструкций. Это оказывается целесообразным при решении ряда задач, к одной из которых относится статический анализ кода. Подобные решения позволяют выделить в специализированные программные объекты низкоуровневый фрагменты кода, отдельно акцентировав на них анализ и возможные преобразования в более надежный код. В работе рассмотрено отображение на отдельный объект-память языка EO низкоуровневой памяти языка C. Вместе с тем ряд аналогичных преобразований требуют дальнейших исследований. В частности, к ним относятся: использование указателей на функции, операторов прямого безусловного перехода, безусловных переходов через указатели на метки и другие. Решение этих задач позволит более формально решать задачи статического анализа.

References

[1] Y. Bugayenko, *Elegant Objects*. [Online]. Available: https://www.elegantobjects.org/.

[2] Y. Bugayenko, *Elegant Objects*. California, USA: Amazon Kindle Direct Publishing, 2016, vol. 1, ISBN: 9781519166913.

[3] Y. Bugayenko, *Elegant Objects*. California, USA: Amazon Kindle Direct Publishing, 2017, vol. 2, ISBN: 9781534908307.

[4] Y. Bugayenko, *EOLANG and φ-calculus*, 2021. arXiv: 2111.13384 [cs.PL].

[5] H. Saleh, S. Zykov, and A. Legalov, «Eolang: Toward a New Java-Based Object-Oriented Programming Language», in *Intelligent Decision Technologies. Smart Innovation, Systems and Technologies*, vol. 238, 2021, pp. 355–363. DOI: 10.1007/978-981-16-2765-1_30.

[6] H. Saleh, J. Attakorah, S. Zykov, and A. Legalov, «Exploring the Eolang-Java Integration and Interoperability», in *Procedia Computer Science. Proceedings of the 25th International Conference KES2021: Knowledge-Based and Intelligent Information & Engineering Systems*, vol. 192, 2021, pp. 4560–4569. DOI: 10.1016/j.procs.2021.09.234.

[7] N. Kudasov and V. Sim, «Formalizing φ-calculus: a purely object-oriented calculus of decorated objects», in *Proceedings of the ECOOP Workshop on Formal Techniques for Java-like Programs (FTfJP 2022)*, Berlin, Germany, 2022.

[8] *Polystat Project: Polyglot Static Analyzer for Object-Oriented Programming Languages*. [Online]. Available: https://github.com/polystat.

[9] Y. Bugayenko, *Reducing Programs to Objects*, 2021. arXiv: 2112.11988 [cs.PL].

[10] *Oberon 07 compiler*. [Online]. Available: https://github.com/vladfolts/oberonjs.

[11] *Visual Interactive Simulations for Education and Science*. [Online]. Available: https://visual.sfu-kras.ru/.

[12] *JSSpeccy 3 a ZX Spectrum emulator in the browser*. [Online]. Available: http://jsspeccy.zxdemo.org/.

[13] *JSNES A JavaScript NES emulator*. [Online]. Available: https://jsnes.org/.

[14] *JSLinux*. [Online]. Available: https://bellard.org/jslinux/.

[15] *Experimental compiler of C to EO*. [Online]. Available: https://github.com/polystat/c2eo.

[16] *Clang: a C language family frontend for LLVM*. [Online]. Available: https://clang.llvm.org/.

[17] *Tutorial for building tools using LibTooling and LibASTMatchers*. [Online]. Available: https://clang.llvm.org/docs/LibASTMatchersTutorial.html.