DOI 10.37882/2223-2966.2025.05.12

ОЦЕНКА СТАТИЧЕСКИХ АНАЛИЗАТОРОВ КОДА GOLANG

EVALUATION OF STATIC ANALYZERS OF GOLANG CODE

O. Evdoshenko

Summary. The article provides an overview the static analyzers of code: golangci-lint, svace, bearer and the Golang. The paper presents templates of regular expression for parsing the output of analyzers. The fields that can be extracted from the outputs of static analyzers are described. The number of unique errors in the analyzers before and after processing was revealed. The criteria for filtering analyzers are described.

Keyword: Golang, static analyzers, regular expression, parsing, syntax error.

Евдошенко Олег Игоревич
кандидат технических наук, доцент, МИРЭА — Российский технологический университет (г. Москва)
evdoshenko@mirea.ru

Аннотация. В статье дано общее представление об статических анализаторах кода: golangci-lint, svace, bearer и языке Go. В работе представлены шаблоны регулярных выражений для парсинга вывода анализаторов. Описаны поля, которые можно извлечь из выводов статических анализаторов. Выявлено количество уникальных ошибок у анализаторов до и после обработки. Описаны критерии фильтрации анализаторов.

Ключевые слова: Golang, статистический анализатор, регулярное выражение, парсинг, синтаксическая ошибка.

Актуальность исследования

Актуальность исследования обуславливается тем, что в современной разработке все больше ценится скорость разработки и быстрый выход на рынок с программным обеспечением, но при этом разрабатываемый программный продукт должен быть безопасен, поскольку нетривиальное поведение программы и проблемы с безопасностью могут повлечь убытки превышающие выгоды от быстрой разработки. К тому же проверка кода на безопасность очень трудоемкий процесс для разработчиков, далеко не всегда приносящий необходимый результат. В связи с этим возникла потребность в автоматической проверке программного обеспечения.

Цель данного исследования — оценить необходимые условия для анализа и провести общую оценку анализаторов.

Общее представление об анализаторах кода и языке Go

Одним из подходов к автоматической проверке кода является статический анализ кода. Статический анализа кода обладает рядом преимуществ, такие как масштабируемость, отсутствие необходимости запускать программное обеспечение на тестовых входных данных, которые делают его довольно эффективным при разработке программного обеспечения. Но также статический анализ кода имеет ряд недостатков, возникающих из принципиально невозможности определить понятие алгоритм и его свойств, которые не позволяют однозначно определить является инструмент для статического анализа кода полезным при использовании в разработке или нет, поэтому для оценки статических анализаторов языка Go требуется описать методику для выявления наилучший решений для статического анализа кода, которая с определенной степенью достоверности оценивает полезные свойства инструментов для статического анализа кода.

Также поскольку язык Go является относительно новым языком, для него еще не выявлены лучшие практики и не созданы признанные инструменты для статического анализа, возникает необходимость в исследовании готовых решений для статического анализа кода Golang, их оценки и выявления наиболее эффективных статических анализаторов, границ их применения и недостатков, а также создания программного продукта для статического анализа кода, дополняющего наиболее перспективные для применения анализаторы и перекрывающий часть их недостатков.

golangci-lint — это мощный инструмент для статического анализа кода на языке программирования Go. Он объединяет множество линтеров, позволяя разработчикам находить и исправлять потенциальные проблемы в коде, улучшая его качество и поддерживаемость.

svace — представляет собой инструмент статического анализа, который используется для выявления уязвимостей в исходном коде программных приложений. Он предоставляет разработчикам и командам безопасности мощные средства для обнаружения потенциально опасных паттернов и уязвимостей в коде до его компиляции или выполнения.

Статистический анализатор bearer обычно используется для оценки и анализа данных, связанных с использованием токенов доступа в системах аутентификации и авторизации.

Описание необходимых условий для анализа и общая оценка анализаторов

Для дальнейшей оценки статических анализаторов недостаточно просто ручной обработки выводов статических анализаторов. Требуется написать обработчик выводов исследуемых статических анализаторов для дальнейшей машинной обработки данных [1].

Обработчики выводов будут представлять из себя лексеры для выделения токенов и преобразователи токенов в нужный формат данных. Формат данных для обработки выбран CSV.

Для написания лексера для golangci-lint будет использоваться генератор кода конечных автоматов [2] flex, а для преобразования данных в формат CSV будет написана программа на C++, а для анализа выводов svace и bearer будут написаны программы преобразователи на основе регулярных выражений.

Для обработки выводов анализаторов требуется тем или иным способом пройти конечным автоматом [2] по тексту вывода и на основе регулярных выражений выделить токены отвечающие за определенные данные в выводе.

Регулярные выражения для парсинга вывода golangci-lint могут иметь следующий шаблон: \\.go:\\d+ (имя файла), «^.*?\\.go» (номер строки с ошибкой), «:\\d+» (колонка, где найдена ошибка), «: .*\\(« (сообщение об ошибке), \\(.*?\\)» (анализатор указавший на ошибку).

Регулярные выражения для парсинга вывода svace могут иметь следующий шаблон: `^.*:` (сообщение об ошибке), `: .*` (описание ошибки), `:\d*$` (номер строки с ошибкой), `/.*\.go:` (имя файла).

Регулярные выражения для парсинга вывода bearer могут иметь следующий шаблон: `^.*: (уровень ошибки, сообщение об ошибке), `:.*:` (имя файла), `^ \d* ` (номер строки, содержащей ошибку).

Далее для получения формата CSV требуется вывести подряд все токены, с используемым разделителем, для CSV разделителем может быть любой символ, либо последовательный набор символов, для отображения принадлежности к полю, добавляется заголовочная строка, отображающая имена полей.

Поля, которые в большинстве случаев можно извлечь из выводов статических анализаторов представляются следующим образом:

- имя файла, в котором зафиксирована ошибка;
- строка, в которой обнаружена ошибка;
- колонка, в которой найдена ошибка;
- уровень значимости ошибки;
- правило, которое обнаружило ошибку или название анализатора, нашедшего ошибку, в случае использования агрегатора;
- словесное описание ошибки;
- локальные строки кода, в которых обнаружена ошибка.

После получения преобразованных данных в формате CSV их можно машинных образом обработать и получить общие оценки анализаторов.

Рассмотрим пересечение ошибок анализаторов. Для этого представим все уникальные срабатывания анализаторов как множество и посмотрим, как пересекаются множества разных анализаторов.

В результате можно выявить количество уникальных ошибок у анализаторов:

- svace — 94;
- bearer — 52;
- golangci-lint — 851 (из них пересечение с bearer — 10).

После просмотра ошибок и построения отношения срабатываний становится очевидных, что многие анализаторы, включенные в golangci-lint, являются либо субъективными, либо срабатывают слишком часто, выдавая слишком много ложных срабатываний. Очевидно, что это исказит результаты дальнейшего исследования. Чтобы это предотвратить требуется подавить неэффективные анализаторы внутри golangci-lint.

Для того чтобы подавить неэффективные анализаторы нужно пройтись по всем ошибкам и выявить те анализаторы, которые очень часто ошибаются и при этом не выдают релевантные ошибки.

После просмотра ошибок, выявляется недостаток анализаторы golangci-lint: в нем очень много анализаторов, которые нацелены на соблюдение некоторых субъективных стандартов кодирования, все такие анализаторы необходимо подавить.

Также подавляются общие ошибки golangci-lint и bearer с целью убрать пересечение и работать только с уникальными ошибками для выявления специализации анализаторов. Подавляться они будут в пользу bearer, поскольку, если подавить их в пользу golangci-lint окажется, что исчезнут многие уникальные ошибки, выявленные только bearer, что уже само по себе говорит о том, что bearer больше специализируется на них и лучше с ними работает. Напротив, golanci-lint ничего не теряет от такого подавления, кроме этих общих ошибок.

После подавления отношение приобретает совершенно другой вид.

Количество уникальных ошибок у анализаторов:
— svace — 94;
— bearer — 51;
— golangci-lint — 72 (из них пересечение с bearer — 1).

Теперь отношение удовлетворяет представлениям о том, как должны выглядеть релевантные срабатывания анализаторов и в каких пропорциях должны находиться. Также можно сделать вывод, что анализаторы имеют практически нулевое пересечение, из чего следует вывод, что они специализированы на разных аспектах анализа кода программы.

ЛИТЕРАТУРА

1. Меньшиков М.А. Обзор моделей работы статических анализаторов // Труды Института системного программирования РАН. — 2021. — №33. — С. 27–40.
2. Хопкрофт Д., Motwani R., Ульман Д. Введение в теорию автоматов, языков и вычислений. — 1-е изд. — СПб.: Вильяме, 2008. — 528 с.
3. Афанасьев В.О., Дворцова В.В., Бородин А.Е. Статический анализатор для языков с обработкой исключений // Труды Института системного программирования РАН. — 2022. — №34. — С. 7–28.
4. Gerasimov A.Yu., Kanakhin A.A., Prilov P.A., Zhukov A.A., Kaminskii E.A. Case study: Source code static analysis for performance issues detection // Proceedings of the Institute for System Programming of the RAS. — 2022. — №34. — С. 7–20.
5. Ilyin D.V., Fokina N.Yu., Semenov V.A. Static dependency analysis for semantic data validation // Proceedings of the Institute for System Programming of the RAS. — 2018. — №30. — С. 271–284.

Журнал «Современная наука: актуальные проблемы теории и практики»