Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 21.Б07-мм

Доработка линтера для OCaml

Хечнев Семен Евгеньевич

Отчёт по учебной практике в форме «Производственное задание»

Научный руководитель:
ассистент кафедры системного программирования Д. С. Косарев

Санкт-Петербург
2023

Оглавление

Введение

Программисты нередко пишут плохой код. Плохой код — это код, который не соответствует принятым нормам в сообществе языка. Для нахождения плохого кода используют линтеры — программы, которые проверяют код на соответствие стандартам в соответствии с определённым набором линтов — правил, описывающих различные конструкции языка. Каждое отдельное правило кажется не очень важным, однако соблюдение всех правил — это база хорошего кода.

Основная задача линтера — сделать код не только единообразным, легко воспринимаемым самим программистом и другими людьми, которые будут изучать код, но и указать программисту на использование методик, которые могут значительно повысить производительность кода.

Zanuda — это линтер для OCaml, разрабатываемый Д. С. Косаревым. Этот линтер имеет пока небольшое количество линтов, но уже активно применяется для проверки домашних заданий у студентов.

Дата сборки: 1 июня 2023 г.

1. Постановка задачи

Целью работы является доработка Zanuda — линтера для OCaml. Для её выполнения были поставлены следующие задачи:

- сделать обзор аналогов;
- реализовать новые линты;
- интегрировать линтер с GitHub Actions;
- выполнить тестирование.

2. Реализация

2.1. Принцип работы Zanuda

Линтер реализован как консольное приложение. На вход линтеру подаётся путь до директории с проектом, в котором должна использоваться система сборки dune. Важно, чтобы исследуемый линтером проект использовал систему сборки dune, так как при помощи неё линтер получает информацию о проекте, а именно пути до файлов в сборке проекта, которые содержат типизированные деревья модулей исследуемой программы. После на этих деревьях запускаются анализы (линты). Линт представляет из себя поиск в дереве определенных паттернов кода.

2.2. Линт о сопоставлении с кортежем

Данный линт находит в коде попытку «распаковки» кортежа при помощи pattern-matching.

Листинг 1: Конструкция, которую находит линт

```
match scru with
| (a,b) → rhs
```

Листинг 2: Конструкция, на которую предлагается заменить конструкцию на Листинге 1

```
let (a, b) = scru in rhs
```

2.3. Линт о взаимно рекурсивных типах

В OCaml для определения взаимно рекурсивных типов используется ключевое слово «and». Однако часто программисты используют это слово там, где взаимной рекурсии нет. Неуместное использование этого ключевого слова может запутать программиста, который будет читать код.

Листинг 3: Пример некорректного использования ключевого слова «and»

```
type t1 = A of t2
and t2 = B
```

Листинг 4: Пример корректного определения типов для Листинга 3

```
type t2 = B
type t1 = A of t2
```

Для реализации этого линта производился обход типов, связанных ключевым словом «and», и построение графа, у которого вершины — это идентификаторы типов, а ребро из некоторой вершины A в вершину B есть только тогда, когда тип A использует в конструкторах или в полях тип B. Тогда типы (вершины), принадлежащие одной компоненте сильной связности этого графа, являются взаимно рекурсивными.

Рис. 1: Граф, описывающий связь типов на Листинге 3

Также важно сообщить пользователю не только типы, которые нужно определять без ключевого слова «and», но и корректный порядок определений этих типов, чтобы исключить ситуацию, при которой линтер советует сначала определить тип A, который использует тип B, а затем уже тип B (при такой декларации типов будет ошибка компиляции, так как тип A использует тип B, но тип B ещё не был определён).

Для нахождения компонент сильной связности использовался алгоритм Косарайю[3]. Важное замечание: этот алгоритм находит компоненты сильной связности в порядке, соответствующем топологической сортировки конденсации исходного графа, поэтому использования дополнительной топологической сортировки конденсации графа для корректного ответа линтера не требуется. В конце линтер добавляет в от-

чёт компоненты сильной связности графа (корректные взаимно рекурсивные декларации типов) в обратном порядке.

Листинг 5: Пример неуместного использования ключевого слова «and». Здесь типы t3 и t4 являются взаимно рекурсивными, остальные же таковыми не являются

```
type t1 = A of t2
and t2 = B of t3
and t3 = C of t4
and t4 = D of t3
```

Листинг 6: Вывод линтера для кода Листинга 5

```
File "bin/main.ml", lines 1–4, characters 0–16:
1 | type t1 = A of t2
2 | and t2 = B of t3
3 | and t3 = C of t4
4 | and t4 = D of t3
Alert zanuda-linter: Unneeded mutual recursion detected in these
type declarations. It's recommended to rewrite 't3', 't4', 't2'
as follows:
                    type t3 =
                      | C of t4
                    and t4 =
                      | D of t3
                    type t2 =
                      | B of t3
```

2.4. Интеграция с GitHub Actions

Необходимо было сделать так, чтобы при добавлении Pull Request автоматически создавался review предлагаемого кода на основе сообщений линтера.

Основная сложность при реализации заключалась в особом способе передачи GitHub API номера строки, которую необходимо прокоммен-

тировать. Согласно документации GitHub API, для комментирования определённой строки в файле необходимо найти номер этой строки относительно первого заголовка чанка этого файла в diff Pull Request'а. Подробнее о формате diff можно прочитать тут[2].

Листинг 7: Нумерация строк в diff. Например, если хотим прокомментировать пятую строчку в обновлённом файле, то необходимо будет передать в API 12 строчку

```
diff --git a/test.js b/test.js
index 2aa9a08..066fc99 100644
--- a/test.js
+++ b/test.js
@@ -2,14 +2,7 @@
                                                                    1
 var hello = require('./hello.js');                                 2
                                                                    3
-var names = [                                                      4
-  'harry',                                                         5
-  'barry',                                                         6
-  'garry',                                                         7
-  'harry',                                                         8
-  'barry',                                                         9
-  'marry',                                                        10
-];                                                                11
+var names = ['harry', 'barry', 'garry', 'harry', 'marry'];       12
                                                                   13
 var names2 = [                                                    14
   'harry',                                                        15
@@ -23,9 +16,7 @@ var names2 = [                                                   16
 // after this line new chunk will be created                      17
 var names3 = [                                                    18
   'harry',                                                        19
-  'barry',                                                        20
-  'garry',                                                        21
```

```
    'harry',                                    22
    'barry',                                    23
-   'marry',                                    24
+   'marry', 'garry',                           25
  ];                                            26
```

Значит для реализация такого инструмента необходимо было реализовать парсер diff. Парсер diff уже был в не совсем законченном состоянии реализован моим научным руководителем. Однако парсер требовал доработки.

Что было сделано:

- Реализована обработка особых случаев в diff. Например, «\No newline at end of file» в конце чанка, «@@ -1 +1 @@» (эквивалентно «@@ -1,1 +1,1 @@»).
- Реализованы тесты для парсера.
- Добавлена функция нумерации строчек в diff.
- Реализовано преобразование сообщений линтера в формат требуемый GitHub API.

В конечном итоге доработан инструмент для создания review. Примеры использования инструмента:

- workflow файл[1];
- Pull request с автоматически сгенерированным review[2].

[1] https://github.com/s-khechnev/demo-review/blob/master/.github/workflows/PrPost.yml (дата доступа: 25 мая 2023 г.).

[2] https://github.com/s-khechnev/demo-review/pull/1 (дата доступа: 25 мая 2023 г.).

3. Тестирование

Для Zanuda уже были реализованы тесты. Тестовое покрытие кода перед началом работы составляло 82%. В процессе работы для каждого добавленного линта и парсера diff также были реализованы тесты. После слияния новых линтов и инструмента для review в основную ветку линтера тестовое покрытие Zanuda осталось прежним.

4. Обзор

4.1. Существующие линтеры

В данном разделе будут рассмотрены различные линтеры, в том числе и линтеры не для OCaml.

4.1.1. Clippy

Clippy[3] — это один из самых популярных линтеров для языка программирования Rust[5].

Особенности:

- около 600 линтов;
- база линтов[4];
- описание каждого линта;
- разделение линтов на уровни и группы;
- возможность настройки конфигурации — отключение некоторых линтов, групп или уровней.

Именно этот линтер стал прообразом для Zanuda. Для Zanuda реализован похожий сайт с базой линтов[5] с описанием каждого линта и разделением линтов на группы и уровни, также в Zanuda есть возможность отключения определенных линтов.

4.1.2. Camelot

Camelot[6] — один из немногих линтеров для OCaml, который минимально поддерживается и развивается.

Особенности:

- расширение для Visual Studio Code[7];

[3] https://github.com/rust-lang/rust-clippy/ (дата доступа: 25 мая 2023 г.).
[4] https://rust-lang.github.io/rust-clippy/master/index.html (дата доступа: 25 мая 2023 г.).
[5] https://kakadu.github.io/zanuda/lints/index.html/ (дата доступа: 25 мая 2023 г.).
[6] https://github.com/upenn-cis1xx/camelot (дата доступа: 25 мая 2023 г.).
[7] https://github.com/esinx/camelot-vscode (дата доступа: 25 мая 2023 г.).

- возможность настройки конфигурации;
- поддержка линтов только для абстрактного синтаксического дерева, что значительно уменьшает количество возможных линтов.

4.1.3. Ocp-lint

Ocp-lint[4] — линтер для OCaml. Особенности:

- наличие DSL — предметно-ориентированного языка для определения новых линтов;
- поддержка линтов не только для абстрактного синтаксического дерева, но и для типизированного дерева, что позволяет реализовать большее количество проверок;
- на данный момент линтер больше не поддерживается[8].

4.2. Использованные технологии

OCaml [1] — язык функционального программирования, который поддерживает императивную, объектно-ориентированную и функциональную парадигмы программирования. OCaml обладает рядом существенных особенностей, которые выделяют его перед другими языками:

- сопоставление с образцом;
- статистическая проверка типов;
- параметрический полиморфизм;
- функции первого класса.

Библиотеки

- Angstrom[9] — парсер-комбинаторы;

[8] https://github.com/upenn-cis1xx/camelot (дата доступа: 25 мая 2023 г.).
[9] https://github.com/inhabitedtype/angstrom (дата доступа: 25 мая 2023 г.).

- yojson[10] — парсер json.

Dune[11] — система сборки для OCaml.

GitHub API — API для взаимодействия с GitHub.

GitHub Actions — инструменты, предоставляющие возможность автоматизировать рабочие процессы разработки программного обеспечения, включая CI/CD.

[10] https://github.com/ocaml-community/yojson (дата доступа: 25 мая 2023 г.).

[11] https://dune.readthedocs.io/en/stable/index.html (дата доступа: 25 мая 2023 г.).

Заключение

В итоге были выполнены все поставленные задачи:

- выполнен обзор существующих линтеров;
- реализовано два линта;
- линтер интегрирован с GitHub Actions;
- выполнено тестирование.

Ссылки

- Линтер — https://github.com/Kakadu/zanuda
- Линт о сопоставлении с кортежем — https://github.com/Kakadu/zanuda/pull/15
- Линт о взаимно рекурсивных типах — https://github.com/Kakadu/zanuda/pull/18
- Инструмент для review — https://github.com/Kakadu/zanuda/pull/22

Список литературы

[1] OCaml. OCaml documentation. — URL: https://ocaml.org/docs (дата обращения: 21 мая 2023 г.).

[2] Wikipedia contributors. Diff — Wikipedia, The Free Encyclopedia. — 2023. — [Online; accessed 31-May-2023]. URL: https://en.wikipedia.org/w/index.php?title=Diff&oldid=1156812942.

[3] Wikipedia contributors. Kosaraju's algorithm — Wikipedia, The Free Encyclopedia. — 2023. — [Online; accessed 31-May-2023]. URL: https://en.wikipedia.org/w/index.php?title=Kosaraju%27s_algorithm&oldid=1143924024.

[4] ocp-lint, A Plugin-based Style-Checker with Semantic Patches / Çagdas Bozman, Théophane Huffschmitt, Michael Laporte, Fabrice Le Fessant // OCaml Users and Developers Workshop 2016. — Nara, Japan, 2016. — . — URL: https://inria.hal.science/hal-01352013.

[5] Википедия. Rust (язык программирования) — Википедия, свободная энциклопедия. — 2022. — [Онлайн; загружено 29 мая 2023]. URL: https://ru.wikipedia.org/?curid=3843010&oldid=127235587.