Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 22.Б15-мм

Поддержка линтера для OCaml

Немакин Никита Антонович

Отчёт по учебной практике
в форме «Решение»

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

Санкт-Петербург
2023

Оглавление

Введение

Когда разработчики следуют идиоматичным и общепринятым практикам при написании программ на любом языке программирования, код становится более предсказуемым. Это во многом упрощает сопровождение и разработку, поскольку написанный код становится легче воспринимать и вносить в него изменения. Итак, следование общепринятым правилам помогает создавать единообразный код, что упрощает его поддержку.

Линтер — инструмент статического анализа исходного кода — становится незаменимым помощником программиста в этом деле, обеспечивая не только высокий стандарт кодирования, но и повышая эффективность и удобство разработки. Он предостерегает разработчика от распространенных ошибок, например, некорректного использования языковых конструкций или попыток переизобрести существующие в языке примитивы.

В мире OCaml примером последнего является Zanuda — изначально разрабатывавшийся в образовательных целях[1] проект, который сегодня имеет достаточно развитую кодовую базу и поддерживает множество диагностических проверок.

В рамках данной работы предлагается реализовать дополнительные линты (проверки) для линтера Zanuda.

[1] https://discuss.ocaml.org/t/how-possible-is-a-clippy-like-linter-for-ocaml/7779/2 (дата доступа: 14 декабря 2023 г.)

Дата сборки: 22 декабря 2023 г.

1. Постановка задачи

Целью работы является расширение функциональности линтера Zanuda, а именно добавление новых проверок для определенных паттернов кода.

Для её выполнения были поставлены следующие задачи:

- Разработать линт об именах конструкторов, встречающихся в стандартной библиотеке языка.
- Разработать линты о реализациях функций `map` и `fold` на списках.
- Протестировать решение.

2. Обзор

В данном разделе рассматриваются технологии, использованные в разработке, а также проводится обзор существующих линтеров для OCaml.

2.1. Инструменты

2.1.1. OCaml

OCaml [3] — функциональный язык программирования общего назначения. Сочетание статической типизации, вывода типов, сопоставления с образцом, алгебраических типов данных и прочих особенностей функциональных языков делает его хорошо подходящим инструментом для задач статического анализа.

Компиляция проекта на OCaml состоит из нескольких этапов. Парсером языка каждая из единиц компиляции, состоящая из модуля и сигнатуры (файлов `.ml` и `.mli` соответственно), преобразуется в дерево абстрактного синтаксиса (AST). Следующая стадия компиляции — вывод и проверка типов — преобразует полученное AST в типизированное, оно содержится в файлах `.cmt` и `.cmti`. Линты Zanuda работают с AST и его типизированной версией.

2.1.2. Dune

Dune[2] — система сборки для OCaml.

2.1.3. Zanuda

Zanuda — линтер для OCaml. Базовый сценарий использования: вызов из консоли с указанием директории скомпилированного с помощью Dune проекта. Благодаря анализу типизированного AST помимо стилистических поддерживаются также и семантические линты. Имеет ряд других особенностей:

[2] https://dune.build/ (дата доступа: 14 декабря 2023 г.)

- общая структура линтов с описанием[3];
- возможность отключения линтов;
- интеграция с GitHub Actions.

2.1.4. Ppxlib

Ppxlib[4] — библиотека для работы с бестиповым AST. Предоставляет возможности для сопоставления с образцом первого класса [1] и создания узлов в AST, а также для обхода AST.

2.2. Обзор аналогов

2.2.1. ocp-lint

Ocp-lint [4] — линтер для OCaml, который также допускает синтаксический и семантический анализ. Большими преимуществами являются конфигурируемость и расширяемость — набор проверок описывается *патчами* — паттернами кода, написанными на собственном предметно-ориентированном языке.

Листинг 1: Пример описания конструкции `if-then-else`, в которой обе ветки совпадают

```
@ConstantIf
expressions: cond, e1, e2
when: "e1 = e2"
```

```
- if cond then e1 else e2
+ ignore (cond:bool); e1
```

К сожалению, проект больше не поддерживается.

[3] https://kakadu.github.io/zanuda/lints/index.html (дата доступа: 14 декабря 2023 г.)
[4] https://github.com/ocaml-ppx/ppxlib (дата доступа: 14 декабря 2023 г.)

2.2.2. Dead code analyzer

Dead code analyzer[5] — проект компании LexiFi, предназначенный для выявления неиспользуемых и недостижимых участков кода программы на OCaml, таких как неиспользуемые импорты, поля классов и конструкторы типов. Также поддерживает небольшой набор синтаксичесих проверок. На данный момент поддержка проекта прекращена.

2.2.3. Camelot

Camelot[6] — линтер, по принципу работы напоминающий Zanuda. Анализирует бестиповые AST, имеет в наличии достаточное число линтов, но почти отсутствуют семантические проверки. Одной из таких является поиск реализаций в коде функций высшего порядка: свёртки и применения для списков. О реализации подобных проверок в Zanuda пойдет речь далее. Имеется расширение для Visual Studio Code. Поддерживается, в отличие от двух предыдущих аналогов.

[5] https://github.com/LexiFi/dead_code_analyzer (дата доступа: 14 декабря 2023 г.)
[6] https://github.com/upenn-cis1xx/camelot (дата доступа: 14 декабря 2023 г.)

3. Реализация

3.1. Неоднозначные имена конструкторов

Если пользовательские типы данных в своем определении содержат конструкторы типов, которые есть в языке, это может привести к конфликту имен в глобальной области видимости. Данный линт ищет пересечения среди имен конструкторов пользовательских типов и типов из стандартной библиотеки языка, а именно `Some` и `None` для типа `option`, а также `Ok` и `Error` для типа `result`.

Листинг 2: Пример конструкции, которую находит линт

```
(* `Some` and `None` are from Stdlib.option *)
type str_option =
  | Some of string
  | None
```

Поиск осуществляется путем анализа типизированного AST, которое даёт информацию о принадлежности типа стандартной библиотеке, а также о том, является ли пользовательский тип псевдонимом для типа из неё. В последнем случае линт не срабатывает.

Листинг 3: В данном случае тип является псевдонимом типа `option`, поэтому линт не срабатывает на нём

```
type 'a maybe = 'a option = None | Some of 'a
```

3.2. `Map` и `fold` для списков

К сожалению, проблемы из области статического анализа неразрешимы в общем случае, т. е. не существует алгоритма, который бы для конкретного кода определял, обладает ли он приписываемым ему свойством или нет. Для конкретной задачи это может быть доказано [2] путём сведения её к неразрешимой проблеме останова. Поэтому, исследуя поведение участка кода программы и не имея при этом никаких дополнительных гарантий, мы можем лишь выдвигать предположения

о его поведении в случае, если он удовлетворяет определенному паттерну.

Решение следующей задачи опирается именно на этот принцип — необходимо детектировать код, который потенциально является реализацией функций `map` или `fold_left` и `fold_right` на списках. Мотивировка следующая — данные функции уже реализованы в стандартной библиотеке языка, и их повторная реализация ухудшает читаемость кода. Кроме того, почти всегда функция `map` реализуется пользователем без использования хвостовой рекурсии, как это сделано в стандартной библиотеке[7], что негативно отражается на производительности.

Листинг 4: Примеры базовых реализаций функций, которые распознаются линтами

```
let rec map f = function
  | [] → []
  | x :: xs → f x :: map f xs

let rec fold_left f acc l = match l with
  | [] → acc
  | x :: xs → fold_left f (f acc a) l

let rec fold_right f l acc = match l with
  | [] → acc
  | x :: xs → f x (fold_right f xs acc)
```

Конечно, обнаружение лишь в точности реализаций, описанных в листинге 4, недостаточно. Необходимо учесть возможность перестановки веток в конструкциях `match` или `function`, число параметров в сигнатуре функции, а также частные случаи реализации. Например, пользователь может определить свёртку для списков из целых чисел, исключив из списка параметров функцию.

Листинг 5: Варианты реализации функций из листинга 4, которые также обнаруживаются линтами

[7] https://v2.ocaml.org/manual/tail_mod_cons.html (дата обращения: 14 декабря 2023 г.)

```
let rec map = function
  | h :: tl → h + 1 :: map1 tl
  | [] → []

let f = (+)
let rec fold_left acc = function
  | x :: xs → fold_left2 (f acc x) xs
  | [] → acc

let sum =
  (* behaviour of fold_right *)
  let rec helper acc = function
    | [] → acc
    | x :: xs → x + helper acc xs
  in helper 0
```

Это достигается путем частичного анализа функции — нас интересует лишь то, что в случае пустого списка возвращается аккумулятор (или пустой список для `map`), а для непустого списка ветвь содержит применение исходной функции к списку без головного элемента. При этом учитывается, что в случае `map` ветвь возвращает список из применения функции к голове и списка из рекурсивного вызова, а для `fold_left` и `fold_right` — соответственно применение исходной или переданной/внешней функции к хвосту списка и аккумулятору.

4. Тестирование

Для каждого реализованного линта были написаны тесты. При этом, несмотря на возможные ложноположительные срабатывания для линтов из пункта 3.2, ложноотрицательных результатов (отсутствие срабатывания, когда это необходимо) в базовых случаях выявлено не было.

Тестовое покрытие (с учётом округления до целого числа процентов) линта об именах конструкторов типов[8] составило 96%, для линтов о реализациях `map`[9] и `fold`[10] — 93%. Общий процент (также с учётом округления) тестового покрытия для всего проекта не изменился.

[8] тестовое покрытие для линта об именах конструкторов (дата обращения: 18 декабря 2023 г.)
[9] тестовое покрытие для линта о `map` (дата обращения: 18 декабря 2023 г.)
[10] тестовое покрытие для линта о `fold` (дата обращения: 18 декабря 2023 г.)

Заключение

В ходе работы были выполнены следующие задачи:

- Разработано два линта:
  - линт об именах конструкторов: https://github.com/Kakadu/zanuda/pull/32;
  - линты о реализациях `fold`/`map` на списках: https://github.com/Kakadu/zanuda/pull/35.
- Проведено тестирование.

Список литературы

[1] Jay Barry, Kesner Delia. First-class patterns // Journal of Functional Programming. — Vol. 19. — ACM, 2009. — P. 191–225. — URL: https://doi.org/10.1017/S0956796808007144 (дата обращения: 17 декабря 2023 г.).

[2] Landi William. Undecidability of static analysis // LOPLAS. — Vol. 1. — ACM, 1992. — P. 323–337. — URL: https://doi.org/10.1145/161494.161501 (дата обращения: 14 декабря 2023 г.).

[3] The OCaml Manual / Xavier Leroy, Damien Doligez, Alain Frisch et al. — 2022. — URL: https://v2.ocaml.org/releases/4.14/htmlman/ (дата обращения: 17 декабря 2023 г.).

[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 (дата обращения: 14 декабря 2023 г.).