Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий
институт

Кафедра «Информатика»
кафедра

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №3

Методология тестирования
тема

Вариант 1

Преподаватель	______________________ подпись, дата	А.С. Кузнецов
Студент	______________________ подпись, дата	В.А. Прекель

Красноярск 2021

1 Цель работы

На конкретных примерах ознакомиться с базовыми методами оценки покрытия блочного тестирования программного обеспечения.

2 Общая постановка задачи

Продемонстрировать понимание и применение на практике ключевых понятий, рассмотренных в этой работе.

3 Ход работы

Для работы был выбран язык OCaml, библиотеки для написания тестов ppx_inline_test, ppx_expect и библиотека для сбора результатов тестирования (coverage) bisect_ppx.

Реализуем тип токена, функцию печати представления токена и функцию парсинга токена:

Листинг 1 – файл lib/parser.ml

```
open Core

type error =
  | NoParensFail
  | OneNumberFail
  | TwoNumberFail
  | TooMany
[@@deriving sexp]

type token =
  | Empty
  | Number of float
  | ParenEmpty
  | ParenOneNumber of float
  | ParenTwoNumbers of float * float
  | OpPlus
  | OpMinus
  | OpMult
  | OpDiv
  | Back
  | Reset
  | Calculate
  | Quit
[@@deriving sexp]

let process_line line =
  let without_parens =
  Option.Let_syntax.(
   let%bind without_prefix = String.chop_prefix ~prefix:"(" line in
   String.chop_suffix ~suffix:")" without_prefix)
 in
 match without_parens with
 | Some l ->
  let splitted = l |> String.split ~on:',' in
  begin
   match splitted with
   | [] ->
    (* String.split cannot return empty list *)
    assert false
   | [ x ] when String.for_all ~f:Char.is_whitespace x -> Ok ParenEmpty
   | [ x ] ->
    begin
     match Caml.Float.of_string_opt x with
     | Some a -> Ok (ParenOneNumber a)
     | None -> Error OneNumberFail
    end
   | [ x; y ] ->
    begin
     match Caml.Float.of_string_opt x, Caml.Float.of_string_opt y with
     | Some a, Some b -> Ok (ParenTwoNumbers (a, b))
     | _ -> Error TwoNumberFail
    end
   | _ -> Error TooMany
  end
 | None ->
  begin
   match line with
   | "" -> Ok Empty
   | "+" -> Ok OpPlus
   | "-" -> Ok OpMinus
   | "*" -> Ok OpMult
   | "/" -> Ok OpDiv
   | "b" | "back" -> Ok Back
   | "c" | "calc" | "calculate" | "=" -> Ok Calculate
   | "r" | "reset" -> Ok Reset
   | "q" | "quit" -> Ok Quit
   | l ->
    begin
     match Caml.Float.of_string_opt l with
     | Some num -> Ok (Number num)
     | None -> Error NoParensFail
    end
  end
;;
```

И задокументируем:

Листинг 2 – файл lib/parser.mli

```ocaml
(** [error] represents the parsing error *)
type error =
  | NoParensFail
  | OneNumberFail
  | TwoNumberFail
  | TooMany
[@@deriving sexp]

(** [token] represents the parsed token *)
type token =
  | Empty
  | Number of float
  | ParenEmpty
  | ParenOneNumber of float
  | ParenTwoNumbers of float * float
  | OpPlus
  | OpMinus
  | OpMult
  | OpDiv
  | Back
  | Reset
  | Calculate
  | Quit
[@@deriving sexp]

(** [process_line str] returns parsed token *)
val process_line : string -> (token, error) result
```

Модуль для проверки на тройное равно:

Листинг 3 – файл lib/tripleorand.ml

```ocaml
open Core

let triple_or (type t) (module E : Equal.S with type t = t) a b c =
  let ( = ) = E.equal in
  a = b || b = c || a = c
;;

let triple_and (type t) (module E : Equal.S with type t = t) a b c =
  let ( = ) = E.equal in
  a = b && b = c && a = c
;;
```

Напишем тесты:

Листинг 4 – фрагмент файла test/test.ml

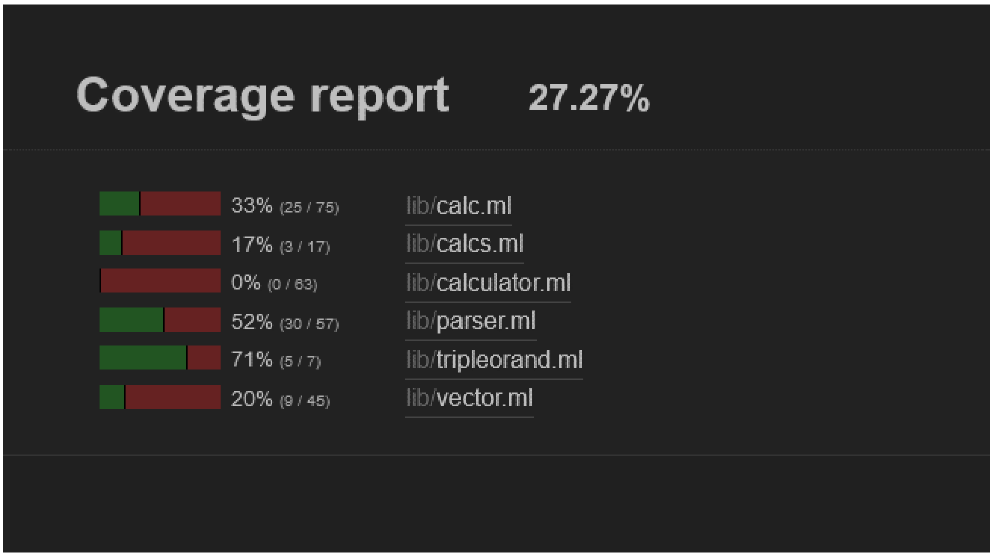
```ocaml
let%test_module "parser" =
  (module struct
    module P = Parser
    let%expect_test "process_line" =
      print_s [%sexp (P.process_line "" : (P.token, P.error) Result.t)];
      [%expect {| (Ok Empty) |}];
      print_s [%sexp (P.process_line "(" : (P.token, P.error) Result.t)];
      [%expect {| (Error NoParensFail) |}];
      print_s [%sexp (P.process_line "()" : (P.token, P.error) Result.t)];
      [%expect {| (Ok ParenEmpty) |}];
      print_s [%sexp (P.process_line "( )" : (P.token, P.error) Result.t)];
      [%expect {| (Ok ParenEmpty) |}];
      print_s [%sexp (P.process_line "(  )" : (P.token, P.error) Result.t)];
      [%expect {| (Ok ParenEmpty) |}];
      print_s [%sexp (P.process_line "(             )" : (P.token, P.error) Result.t)];
      [%expect {| (Ok ParenEmpty) |}];
      print_s [%sexp (P.process_line "(12)" : (P.token, P.error) Result.t)];
      [%expect {| (Ok (ParenOneNumber 12)) |}];
      print_s [%sexp (P.process_line "(123)" : (P.token, P.error) Result.t)];
      [%expect {| (Ok (ParenOneNumber 123)) |}];
      print_s [%sexp (P.process_line "(123.12)" : (P.token, P.error) Result.t)];
      [%expect {| (Ok (ParenOneNumber 123.12)) |}];
      print_s [%sexp (P.process_line "(123q)" : (P.token, P.error) Result.t)];
      [%expect {| (Error OneNumberFail) |}];
      print_s [%sexp (P.process_line "(123,123)" : (P.token, P.error) Result.t)];
      [%expect {| (Ok (ParenTwoNumbers 123 123)) |}];
      print_s [%sexp (P.process_line "(123.12,123)" : (P.token, P.error) Result.t)];
      [%expect {| (Ok (ParenTwoNumbers 123.12 123)) |}];
      print_s [%sexp (P.process_line "(arsa,asrt)" : (P.token, P.error) Result.t)];
      [%expect {| (Error TwoNumberFail) |}];
      print_s
        [%sexp
          (P.process_line "(1231, 123122222222222222222222222222222222221.2121)"
            : (P.token, P.error) Result.t)];
      [%expect {| (Ok (ParenTwoNumbers 1231 1.2312222222222223E+38)) |}];
      print_s
        [%sexp
          (P.process_line "(sntratroaeintstsrdrsatd.r.s,rnetrapl 283;9tp;hawlu)"
            : (P.token, P.error) Result.t)];
      [%expect {| (Error TwoNumberFail) |}];
      print_s [%sexp (P.process_line "(1,2,3)" : (P.token, P.error) Result.t)];
      [%expect {| (Error TooMany) |}]
    ;;
  end)
;;

let%test_module "triple_or and triple_and tests" =
  (module struct
    module TrAO = Tripleorand

    let%test "triple_or v0" = TrAO.triple_or (module V0) V0.make V0.make V0.make
    let%test "triple_or v0" = TrAO.triple_and (module V0) V0.make V0.make V0.make

    let%test "triple_or v1 not" =
      not @@ TrAO.triple_or (module V1) (V1.make 1.) (V1.make 2.) (V1.make 3.)
    ;;

    let%test "triple_or v1 not" =
      not @@ TrAO.triple_and (module V1) (V1.make 1.) (V1.make 2.) (V1.make 3.)
    ;;

    let%test "triple_or v1" =
      TrAO.triple_or (module V1) (V1.make 1.) (V1.make 1.) (V1.make 3.)
    ;;

    let%test "triple_or v1" =
      TrAO.triple_and (module V1) (V1.make 1.) (V1.make 1.) (V1.make 1.)
    ;;

    let print_bool = printf "%b\n"

    let%expect_test "" =
      TrAO.triple_and (module Int) 1 2 3 |> print_bool;
      [%expect {| false |}];
      TrAO.triple_or (module Int) 1 1 3 |> print_bool;
      [%expect {| true |}]
    ;;
  end)
;;
```

Запустим тесты вместе с покрытием и генерацией отчёта:

Листинг 5 – запуск тестов и генерация отчёта

```
vladislav@DESKTOP-NK6MA9B:~/Projects/software-testing$ make coverage
find . -name '*.coverage' | xargs rm -f
opam exec -- dune runtest --instrument-with bisect_ppx --force
bisect-ppx-report html
```

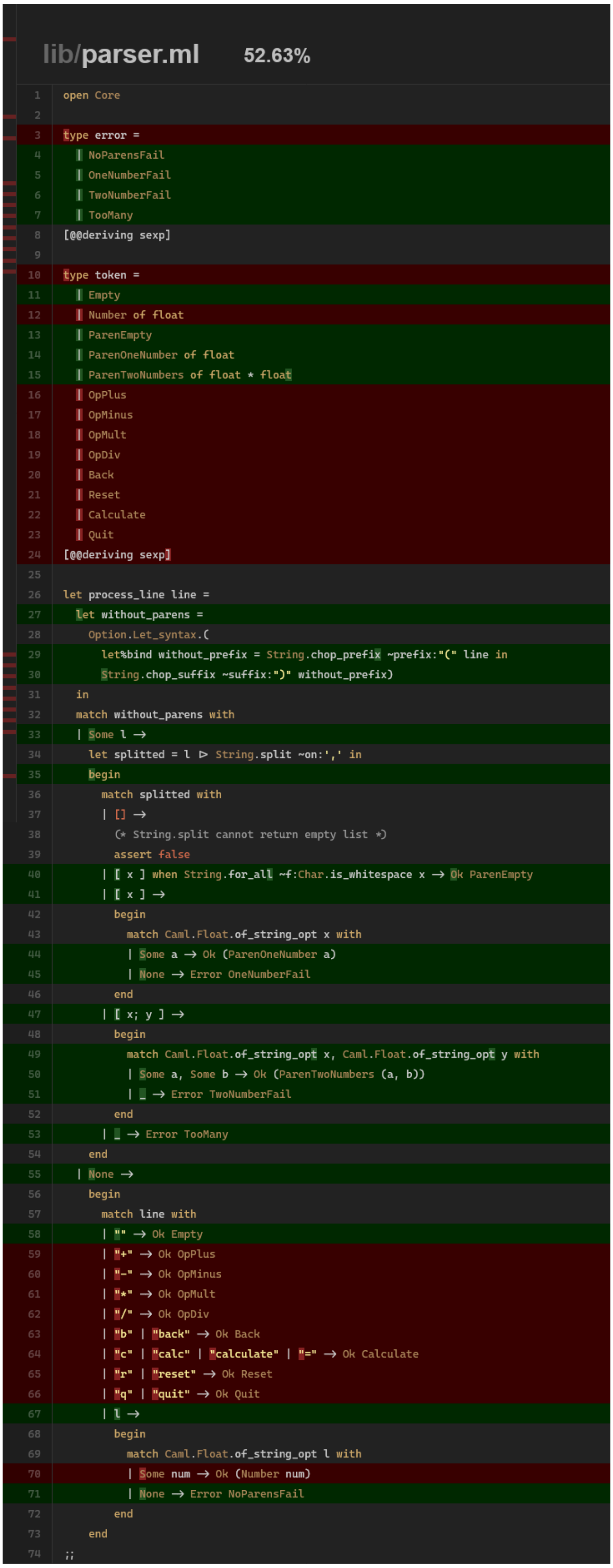
Выполнилось без ошибок – значит тесты прошли успешно. Посмотрим на покрытие:



Coverage report 27.27%

33% (25 / 75) lib/calc.ml
17% (3 / 17) lib/calcs.ml
0% (0 / 63) lib/calculator.ml
52% (30 / 57) lib/parser.ml
71% (5 / 7) lib/tripleorand.ml
20% (9 / 45) lib/vector.ml

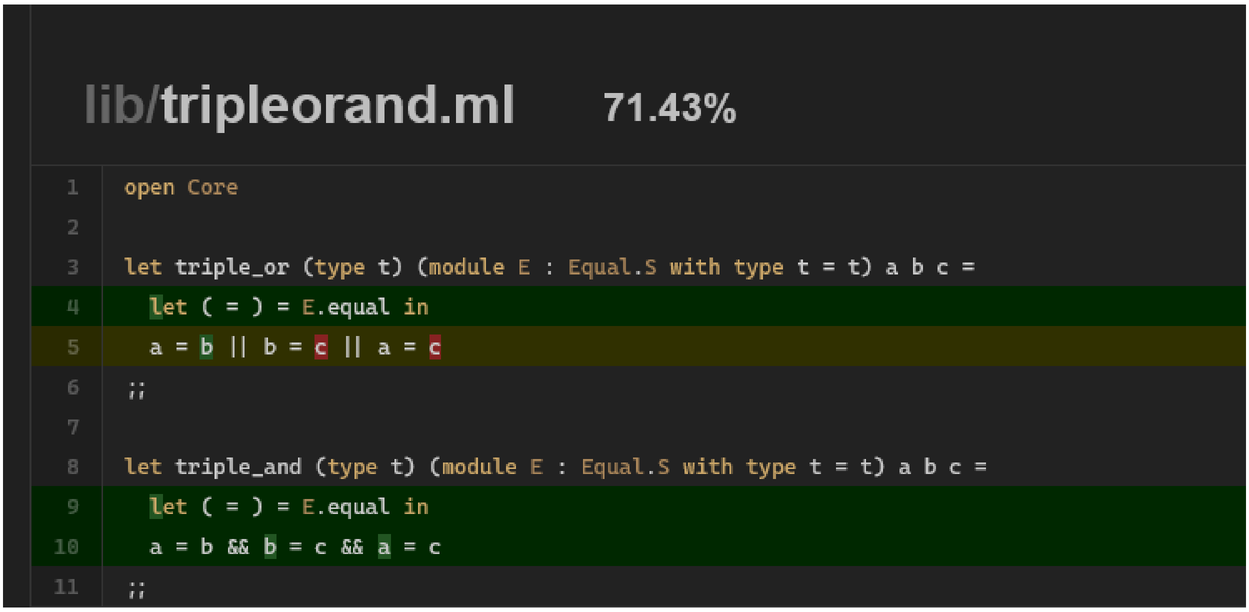
Рисунок 1 – Файлы в отчёте



lib/parser.ml 52.63%

```
open Core

type error =
  | NoParensFail
  | OneNumberFail
  | TwoNumberFail
  | TooMany
[@@deriving sexp]

type token =
  | Empty
  | Number of float
  | ParenEmpty
  | ParenOneNumber of float
  | ParenTwoNumbers of float * float
  | OpPlus
  | OpMinus
  | OpMult
  | OpDiv
  | Back
  | Reset
  | Calculate
  | Quit
[@@deriving sexp]

let process_line line =
  let without_parens =
    Option.Let_syntax.(
      let%bind without_prefix = String.chop_prefix ~prefix:"(" line in
      String.chop_suffix ~suffix:")" without_prefix)
  in
  match without_parens with
  | Some l →
    let splitted = l ▷ String.split ~on:',' in
    begin
      match splitted with
      | [] →
        (* String.split cannot return empty list *)
        assert false
      | [ x ] when String.for_all ~f:Char.is_whitespace x → Ok ParenEmpty
      | [ x ] →
        begin
          match Caml.Float.of_string_opt x with
          | Some a → Ok (ParenOneNumber a)
          | None → Error OneNumberFail
        end
      | [ x; y ] →
        begin
          match Caml.Float.of_string_opt x, Caml.Float.of_string_opt y with
          | Some a, Some b → Ok (ParenTwoNumbers (a, b))
          | _ → Error TwoNumberFail
        end
      | _ → Error TooMany
    end
  | None →
    begin
      match line with
      | "" → Ok Empty
      | "+" → Ok OpPlus
      | "-" → Ok OpMinus
      | "*" → Ok OpMult
      | "/" → Ok OpDiv
      | "b" | "back" → Ok Back
      | "c" | "calc" | "calculate" | "=" → Ok Calculate
      | "r" | "reset" → Ok Reset
      | "q" | "quit" → Ok Quit
      | l →
        begin
          match Caml.Float.of_string_opt l with
          | Some num → Ok (Number num)
          | None → Error NoParensFail
        end
    end
;;
```

Рисунок 2 – Покрытие файла lib/parser.ml



lib/tripleorand.ml 71.43%

```
open Core

let triple_or (type t) (module E : Equal.S with type t = t) a b c =
  let ( = ) = E.equal in
  a = b || b = c || a = c
;;

let triple_and (type t) (module E : Equal.S with type t = t) a b c =
  let ( = ) = E.equal in
  a = b && b = c && a = c
;;
```

Рисунок 3 – покрытие файла lib/tripleorand.ml

Видно, что покрытие не 100-процентное. Допишем тесты.

Напишем заведомо неверный тест. Библиотека ppx_expect сама предложит исправить его так, чтобы он стал верным:

Листинг 6 – фрагмент файла test/test.ml

```
let%expect_test "to 100 percent" =
      print_s [%sexp (P.process_line "+" : (P.token, P.error) Result.t)];
      [%expect {| (Ok Empty) |}]
    ;;
```

Листинг 7 – запуск тестов и исправления теста

```
vladislav@DESKTOP-NK6MA9B:~/Projects/software-testing$ dune runtest
File "test/test.ml", line 1, characters 0-0:
------ test/test.ml
++++++ test/test.ml.corrected
File "test/test.ml", line 51, characters 0-1:
 |      print_s
 |        [%sexp
 |          (P.process_line "(1231,
123122222222222222222222222222222222221.2121)"
 |            : (P.token, P.error) Result.t)];
 |      [%expect {| (Ok (ParenTwoNumbers 1231 1.2312222222222223E+38)) |}];
 |      print_s
 |        [%sexp
 |          (P.process_line "(sntratroaeintstsrdrsatd.r.s,rnetrapl
283;9tp;hawlu)"
 |            : (P.token, P.error) Result.t)];
 |      [%expect {| (Error TwoNumberFail) |}];
 |      print_s [%sexp (P.process_line "(1,2,3)" : (P.token, P.error)
Result.t)];
 |      [%expect {| (Error TooMany) |}]
 |    ;;
 |
 |    let%expect_test "to 100 percent" =
 |      print_s [%sexp (P.process_line "+" : (P.token, P.error) Result.t)];
-|      [%expect {| (Ok Empty) |}]
+|      [%expect {| (Ok OpPlus) |}]
 |    ;;
 |  end)
 |;;
 |
 |module V0 = Vector.Vector0
 |module V1 = Vector.Vector1
 |module V2 = Vector.Vector2
 |
 |let%test_module "vector parsing tests" =
 |  (module struct
 |    let%expect_test "" =
 |      let a = Lab03_parser.Parser.Empty in
 |      print_s [%sexp (a : Lab03_parser.Parser.token)];
 |      [%expect {| Empty |}]
 |    ;;
 |
vladislav@DESKTOP-NK6MA9B:~/Projects/software-testing$ dune promote
Promoting _build/default/test/test.ml.corrected to test/test.ml.
```

Теперь написанный тест выглядит так:

Листинг 8 – фрагмент файла test/test.ml

```
let%expect_test "to 100 percent" =
  print_s [%sexp (P.process_line "+" : (P.token, P.error) Result.t)];
  [%expect {| (Ok OpPlus) |}]
;;
```

Таким образом допишем тесты для 100-процентного покрытия.

Листинг 9 – фрагмент файла test/test.ml

```
let%expect_test "to 100 percent" =
      print_s [%sexp (P.process_line "+" : (P.token, P.error) Result.t)];
      [%expect {| (Ok OpPlus) |}];
```

```
print_s [%sexp (P.process_line "-" : (P.token, P.error) Result.t)];
[%expect {| (Ok OpMinus) |}];
print_s [%sexp (P.process_line "*" : (P.token, P.error) Result.t)];
[%expect {| (Ok OpMult) |}];
print_s [%sexp (P.process_line "/" : (P.token, P.error) Result.t)];
[%expect {| (Ok OpDiv) |}];
print_s [%sexp (P.process_line "b" : (P.token, P.error) Result.t)];
[%expect {| (Ok Back) |}];
print_s [%sexp (P.process_line "back" : (P.token, P.error) Result.t)];
[%expect {| (Ok Back) |}];
print_s [%sexp (P.process_line "c" : (P.token, P.error) Result.t)];
[%expect {| (Ok Calculate) |}];
print_s [%sexp (P.process_line "calc" : (P.token, P.error) Result.t)];
[%expect {| (Ok Calculate) |}];
print_s [%sexp (P.process_line "calculate" : (P.token, P.error) Result.t)];
[%expect {| (Ok Calculate) |}];
print_s [%sexp (P.process_line "=" : (P.token, P.error) Result.t)];
[%expect {| (Ok Calculate) |}];
print_s [%sexp (P.process_line "r" : (P.token, P.error) Result.t)];
[%expect {| (Ok Reset) |}];
print_s [%sexp (P.process_line "reset" : (P.token, P.error) Result.t)];
[%expect {| (Ok Reset) |}];
print_s [%sexp (P.process_line "quit" : (P.token, P.error) Result.t)];
[%expect {| (Ok Quit) |}];
print_s [%sexp (P.process_line "q" : (P.token, P.error) Result.t)];
[%expect {| (Ok Quit) |}];
print_s [%sexp (P.process_line "123" : (P.token, P.error) Result.t)];
[%expect {| (Ok (Number 123)) |}];
print_s [%sexp (P.process_line "-123" : (P.token, P.error) Result.t)];
[%expect {| (Ok (Number -123)) |}];
print_s [%sexp (P.process_line "-0" : (P.token, P.error) Result.t)];
[%expect {| (Ok (Number -0)) |}];
print_s [%sexp (P.process_line "0" : (P.token, P.error) Result.t)];
[%expect {| (Ok (Number 0)) |}];
[%sexp (P.Empty : P.token)] |> P.token_of_sexp |> P.sexp_of_token |> print_s;
[%expect {| Empty |}];
[%sexp (P.Number 123123. : P.token)]
|> P.token_of_sexp
|> P.sexp_of_token
|> print_s;
[%expect {| (Number 123123) |}];
[%sexp (P.ParenEmpty : P.token)] |> P.token_of_sexp |> P.sexp_of_token |> print_s;
[%expect {| ParenEmpty |}];
[%sexp (P.ParenOneNumber 2131. : P.token)]
|> P.token_of_sexp
|> P.sexp_of_token
|> print_s;
[%expect {| (ParenOneNumber 2131) |}];
[%sexp (P.ParenTwoNumbers (12312., 1231.) : P.token)]
|> P.token_of_sexp
|> P.sexp_of_token
|> print_s;
[%expect {| (ParenTwoNumbers 12312 1231) |}];
```

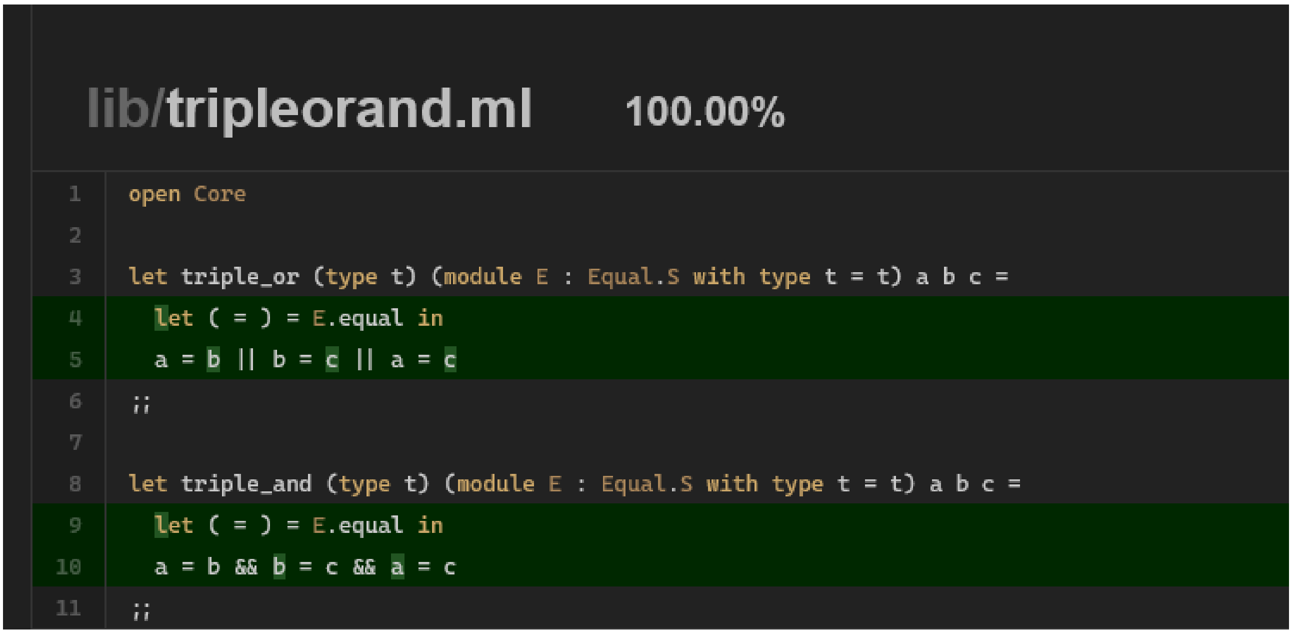
```
   [%sexp (P.NoParensFail : P.error)] |> P.error_of_sexp |> P.sexp_of_error |>
print_s;
   [%expect {| NoParensFail |}];
   [%sexp (P.OneNumberFail : P.error)] |> P.error_of_sexp |> P.sexp_of_error |>
print_s;
   [%expect {| OneNumberFail |}];
   [%sexp (P.TwoNumberFail : P.error)] |> P.error_of_sexp |> P.sexp_of_error |>
print_s;
   [%expect {| TwoNumberFail |}];
   [%sexp (P.TooMany : P.error)] |> P.error_of_sexp |> P.sexp_of_error |> print_s;
   [%expect {| TooMany |}]
  ;;
```

Допишем тесты для тройного равно:

Листинг 10 – фрагмент файла test/test.ml

```
let%expect_test "to 100 percent" =
  TrAO.triple_or (module Int) 2 1 1 |> print_bool;
  [%expect {| true |}];
  TrAO.triple_or (module Int) 0 1 0 |> print_bool;
  [%expect {| true |}]
;;
```

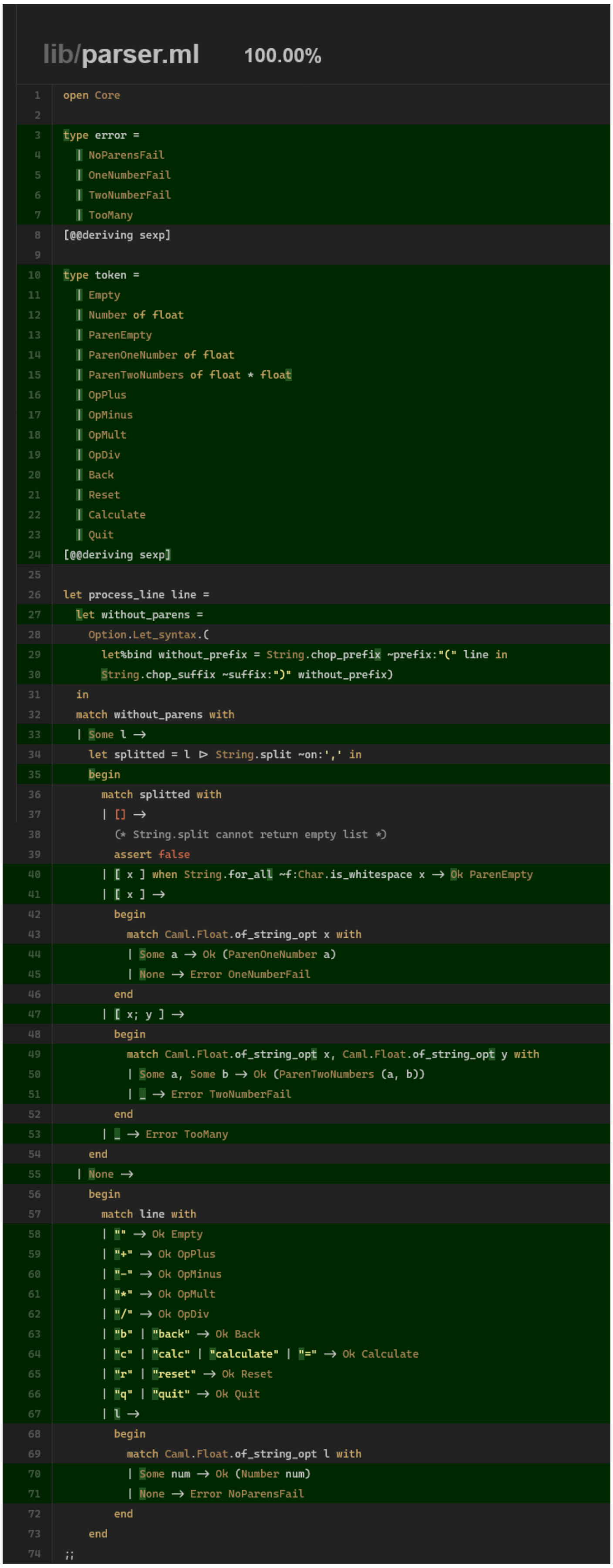
Посмотрим на отчёт:



lib/tripleorand.ml 100.00%

```
open Core

let triple_or (type t) (module E : Equal.S with type t = t) a b c =
  let ( = ) = E.equal in
  a = b || b = c || a = c
;;

let triple_and (type t) (module E : Equal.S with type t = t) a b c =
  let ( = ) = E.equal in
  a = b && b = c && a = c
;;
```

Рисунок 4 – покрытие файла lib/tripleorand.ml



lib/parser.ml 100.00%

```ocaml
open Core

type error =
  | NoParensFail
  | OneNumberFail
  | TwoNumberFail
  | TooMany
[@@deriving sexp]

type token =
  | Empty
  | Number of float
  | ParenEmpty
  | ParenOneNumber of float
  | ParenTwoNumbers of float * float
  | OpPlus
  | OpMinus
  | OpMult
  | OpDiv
  | Back
  | Reset
  | Calculate
  | Quit
[@@deriving sexp]

let process_line line =
  let without_parens =
    Option.Let_syntax.(
      let%bind without_prefix = String.chop_prefix ~prefix:"(" line in
      String.chop_suffix ~suffix:")" without_prefix)
  in
  match without_parens with
  | Some l ->
    let splitted = l |> String.split ~on:',' in
    begin
      match splitted with
      | [] ->
        (* String.split cannot return empty list *)
        assert false
      | [ x ] when String.for_all ~f:Char.is_whitespace x -> Ok ParenEmpty
      | [ x ] ->
        begin
          match Caml.Float.of_string_opt x with
          | Some a -> Ok (ParenOneNumber a)
          | None -> Error OneNumberFail
        end
      | [ x; y ] ->
        begin
          match Caml.Float.of_string_opt x, Caml.Float.of_string_opt y with
          | Some a, Some b -> Ok (ParenTwoNumbers (a, b))
          | _ -> Error TwoNumberFail
        end
      | _ -> Error TooMany
    end
  | None ->
    begin
      match line with
      | "" -> Ok Empty
      | "+" -> Ok OpPlus
      | "-" -> Ok OpMinus
      | "*" -> Ok OpMult
      | "/" -> Ok OpDiv
      | "b" | "back" -> Ok Back
      | "c" | "calc" | "calculate" | "=" -> Ok Calculate
      | "r" | "reset" -> Ok Reset
      | "q" | "quit" -> Ok Quit
      | l ->
        begin
          match Caml.Float.of_string_opt l with
          | Some num -> Ok (Number num)
          | None -> Error NoParensFail
        end
    end
;;
```

Рисунок 5 – покрытие файла lib/parser.ml

4 Вывод

В данной работе мы ознакомились с основами тестирования с покрытием кода на языке OCaml с применением библиотек ppx_inline_test, ppx_expect и bisect_ppx. Исходный код доступен на GitHub.