KØBENHAVNS UNIVERSITET

Home Assignment 3

A Boa Parser-Haskell

qbp758 / zxf339

AP22 Assignment 3 Group 10

Date: 30. september 2022

Part 1: Document and Explain Non-trivial modifications

***1. How we dealt with ambiguity and/or left-recursion, biased combinators (<++, munch)

Regarding the $left\ recursion$, we find one $left\ recursion$ in *Expr* expression as follows.

$$Expr \quad ::= \quad Expr \quad Oper \quad Expr$$

Thus, we eliminate the left recursion in *Expr* and the *Expr*, *Oper* syntax of Boa as below.

$$Expr ::= not \;\; Expr \mid Expr'$$

$$Expr' ::= Expr'' \;\; LExpr \mid Expr''$$

$$LExpr ::= LOper \;\; Expr'' \;\; LExpr \mid COper \;\; Expr'' \;\; LExpr$$

$$LOper ::= ArithOper \mid FactOper$$

$$ArithOper ::= + \mid -$$

$$FactOper ::= * \mid // \mid$$

$$COper ::= \;\; == \;\; \mid\ != \;\mid\; < \;\mid\; <= \;\mid\; > \;\mid\; >= \;\mid in \mid not \;\; in$$

About $< ++$ combinators, we use this to implement $left - associative$, the code as follows.

```haskell
-- LOper = ArithOper | FactOper
lOp :: ReadP String
lOp =
    arithmeticOp <++ factOp
```

About *munch* and $munch1$, we use this to parse constant numbers. Code snippet is as follows.

```haskell
-- parse constant number
endC <- munch1 isDigit
```

Part 2: Assessment of the Quality of Our code

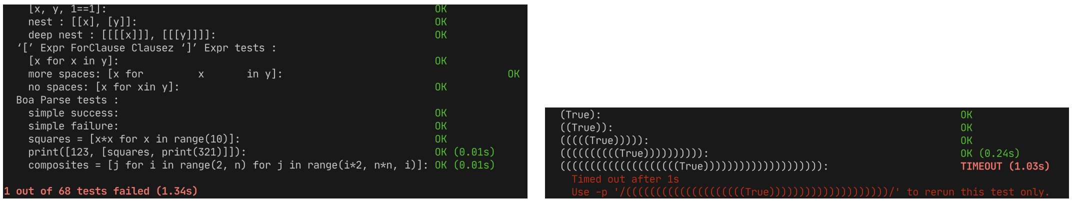
A. Completeness

We have completed all the questions, although we spent a bunch of time, way more than last time.

B. Correctness

Our code has almost passed the unit-testing (1 out of 68 failed) and the onlineTA-testing (9 out of 111 failed).

Figur 1: Result of Unit Testing



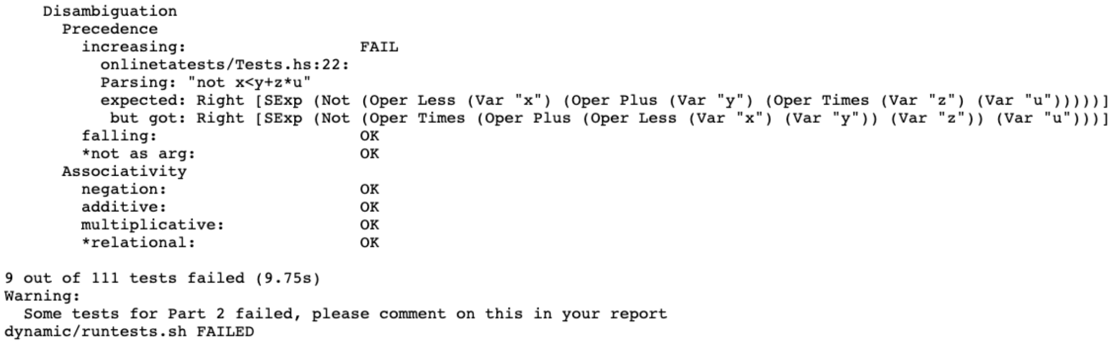
```
    [x, y, 1==1]:                                                OK
    nest : [[x], [y]]:                                           OK
    deep nest : [[[[x]]], [[[y]]]]:                              OK
  '[' Expr ForClause Clausez ']' Expr tests :
    [x for x in y]:                                              OK
    more spaces: [x for         x       in y]:                         OK
    no spaces: [x for xin y]:                                    OK
  Boa Parse tests :
    simple success:                                              OK
    simple failure:                                              OK
    squares = [x*x for x in range(10)]:                          OK
    print([123, [squares, print(321)]]):                         OK (0.01s)
    composites = [j for i in range(2, n) for j in range(i*2, n*n, i)]: OK (0.01s)

1 out of 68 tests failed (1.34s)
```

(a)

```
    (True):                                                      OK
    ((True)):                                                    OK
    (((((True))))):                                              OK
    ((((((((((True)))))))))):                                    OK (0.24s)
    ((((((((((((((((((((True)))))))))))))))))))):                TIMEOUT (1.03s)
      Timed out after 1s
      Use -p '/((((((((((((((((((((True))))))))))))))))))))/' to rerun this test only.
```

(b)

Figur 2: Result of OnlineTA Testing



```
    Disambiguation
      Precedence
        increasing:                FAIL
          onlinetatests/Tests.hs:22:
          Parsing: "not x<y+z*u"
          expected: Right [SExp (Not (Oper Less (Var "x") (Oper Plus (Var "y") (Oper Times (Var "z") (Var "u")))))]
           but got: Right [SExp (Not (Oper Times (Oper Plus (Oper Less (Var "x") (Var "y")) (Var "z")) (Var "u")))]
        falling:                   OK
        *not as arg:               OK
      Associativity
        negation:                  OK
        additive:                  OK
        multiplicative:            OK
        *relational:               OK

9 out of 111 tests failed (9.75s)
Warning:
  Some tests for Part 2 failed, please comment on this in your report
dynamic/runtests.sh FAILED
```

Details about 9 errors of onlineTA. There are 8 timeout cases and 1 fail case. We see the timeout cases, We found these timeout cases mainly occur in deep parenthesis and deep brackets. The deep brackets lead timeout also occurs in our unit tests. In unit tests, We found that timeout occurs when there are about 20 brackets or more. We want to fix it but we do not have enough time and we did not find what caused the timeout.
Regarding the one fail case, we fixed it but then it causes more bugs. We are confused about this bug, so we are going to ask the TA in the lab class next week.

C. Efficiency

We did try our best to improve the efficiency of code operation and make efforts to improve space usage, but it turns out that not much progress was made in our efforts.

***D. Robustness

The Boa interpreter we designed can cope with errors and erroneous input. We have implemented some errors processing, for example, in the ‘div‘ ‘Mod‘ operation, and it will report an error message when the dividend number or modulus equals zero; For erroneous input, in the ‘in‘ and ‘list comprehension‘ operations, if the parameter is non-list, it will report an error message.

E. Maintainability

We both think the maintainability of our code is quite good. As we added enough common code to improve the readability of the code and abstracted functions to make our code logic clarified.