Name: Chakrya Ros

Project Report : Parser Lettuce

I decided to implement Parse Lettuce for this project. Firstly, I am doing research about parsing. I read some articles that are useful for me to start my project. I found one article on github that explain how to use parsing combinator library. And I read the book in library, the Programming in Scala. It's also helpful for me to start my project.

After I read those documents, and I started writing the definition and inference rule.

ParserLettuce ::= identifier (Parser[Expr])

| boolParse (Parser[Expr])

| PositiveParser (Parser[Expr])

| NegativeParser (Parser[Expr])

| symbol (Parser[String])

| Logic_Parser (Parser[String])

| If_Parser (Parser[String])

| then_Parser (Parser[String])

| else_Parser (Parser[String])

| Let_Parser (Parser[String])

| in_Parser (Parser[String])

| LetRec_Parser(Parser[String])

| expr(identifier, boolParser, PositiveParser, NegativeParser, eval_let,
eval_FunDef, eval_FunCall, eval_LetRec, eval_logic ,eval_equals)

| eval_Arth(expr, symbol, expr)

| eval_ifThenElse(If_Parser, expr, then_Parser, expr, else_Parser expr)

| eval_Let(Let_Parser, expr, equals, expr, in_Parser, expr)

| eval_FunDef(String, "(", String, ")", expr)

| eval_FunCall("(", expr, "[", expr "]", ")")

| eval_LetRec(LetRec, str, equalsl, str, "(", expr, ")" expr, in_Parser, expr)

| apply(String)

These are inference rules:

$$\frac{i \in [a\text{-}zA\text{-}Z],\ i \Downarrow x}{\text{identifier}(i) \Downarrow \text{Ident}(x)}\ (\text{identifier})$$

$$\frac{e \in [\text{"True"} \mid \text{"False"} \mid \text{"false"} \mid \text{"true"}],\ e \Downarrow p}{\text{boolParser}(e) \Downarrow \text{Bin}(p)}\ (\text{boolParser})$$

$$\frac{i \in [0|1\text{-}9] \Downarrow \text{num},\ \text{Int\_to\_nat}(\text{num})}{\text{PositiveParser}(i) \Downarrow \text{Const}(\text{Positive}(\text{num}))}\ (\text{PositiveParser})$$

$$\frac{i \in [0|1\text{-}9] \Downarrow \text{num},\ \text{Int\_to\_nat}(\text{num})}{\text{NegativeParser}(-i) \Downarrow \text{Const}(\text{Negative}(\text{num}))}\ (\text{NegativeParser})$$

$$\frac{i \in [\text{"+"} \mid \text{"-"} \mid \text{"*"} \mid \text{"^"}],\ i \Downarrow s}{\text{symbol}(e) \Downarrow s}\ (\text{symbol})$$

$$\frac{e_1 \Downarrow \text{Const}(e_1),\ e_2 \Downarrow \text{Const}(e_2),\ s \in \text{Symbol}}{\text{eval\_Arith}(e_1, s, e_2) \Downarrow f(e_1, e_2)}\ (\text{eval\_Arith})$$

where $f_+\ e_1\ e_1 = \text{Plus}(e_1, e_2)$

$f_-\ e_1\ e_2 = \text{Minus}(e_1, e_2)$

$f_*\ e_1\ e_2 = \text{Mult}(e_1, e_2)$

$f_\wedge\ e_1\ e_2 = \text{Pow}(e_1, e_2)$

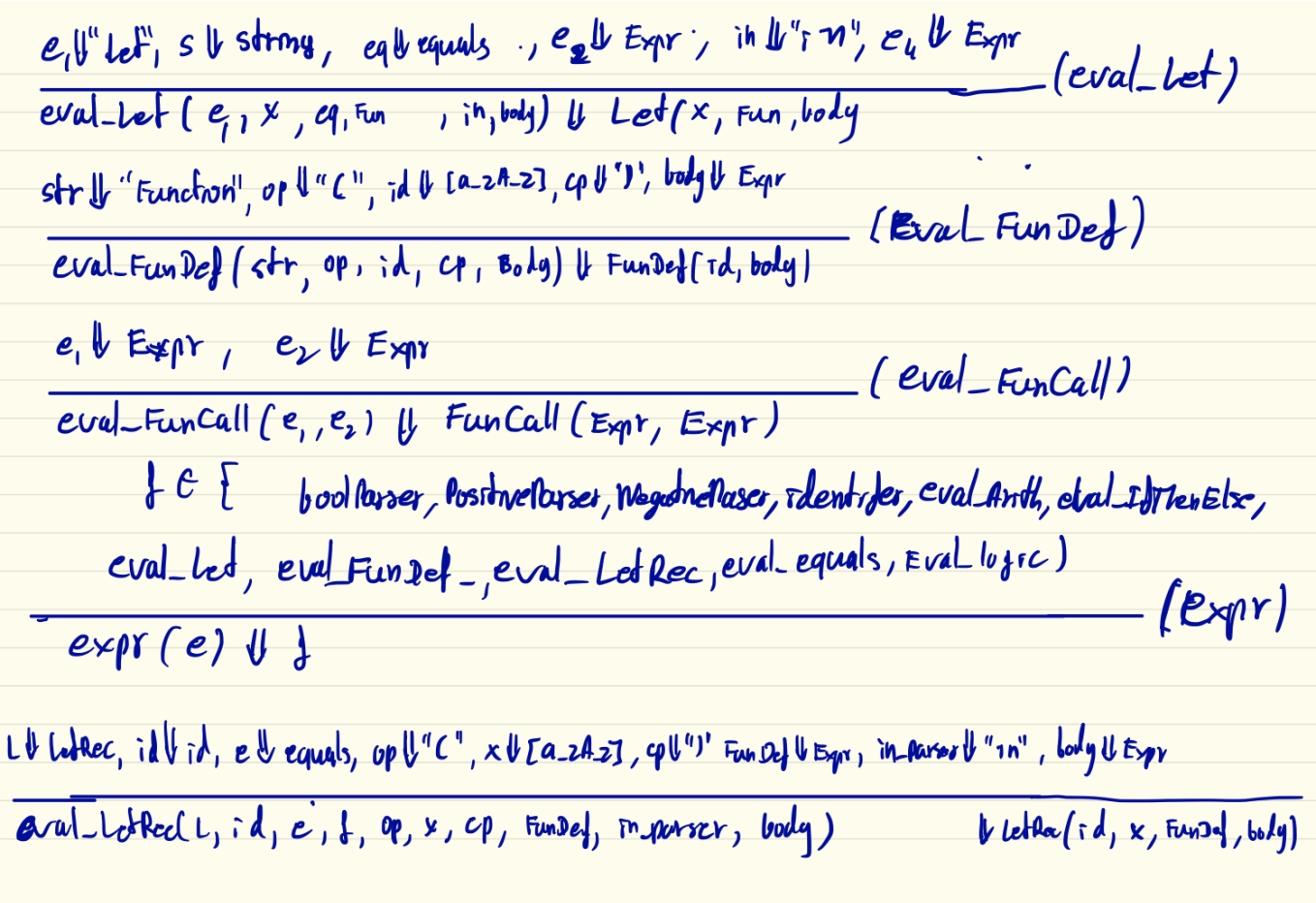
$$\frac{\text{If} \Downarrow \text{"If"},\ x \Downarrow \text{Expr},\ \text{Then} \Downarrow \text{"Then"},\ e_1 \Downarrow \text{Expr},\ \text{Else} \Downarrow \text{"Else"},\ e_2 \Downarrow \text{Expr}}{\text{eval\_IfThenElse}(\text{If}, x, \text{Then}, e_1, \text{Else}, e_2) \Downarrow \text{IfThenElse}(x, e_1, e_2)}\ (\text{eval\_IfThenElse})$$

$$\frac{x \Downarrow \text{Expr},\ y \Downarrow \text{Expr},\ eq \Downarrow \text{"="}}{\text{Eval\_equals}(x, eq, x) \Downarrow \text{Eq}(x, y)}\ (\text{Eval\_equals})$$

$$\frac{x \Downarrow \text{Expr},\ y \Downarrow \text{Expr},\ \text{logic} \Downarrow \text{"and"} \mid \text{"And"} \mid \text{"or"} \mid \text{"OR"},\ f \in \{\text{and}, \text{or}\}}{\text{eval\_logic}(x, \text{logic}, y) \Downarrow f(x, y)}\ (\text{Eval\_logic})$$

where $f_{\text{and}}(x, y) = \text{And}(x, y)$

$f_{\text{or}}(x, y) = \text{Or}(x, y)$



I have learned some symbols like

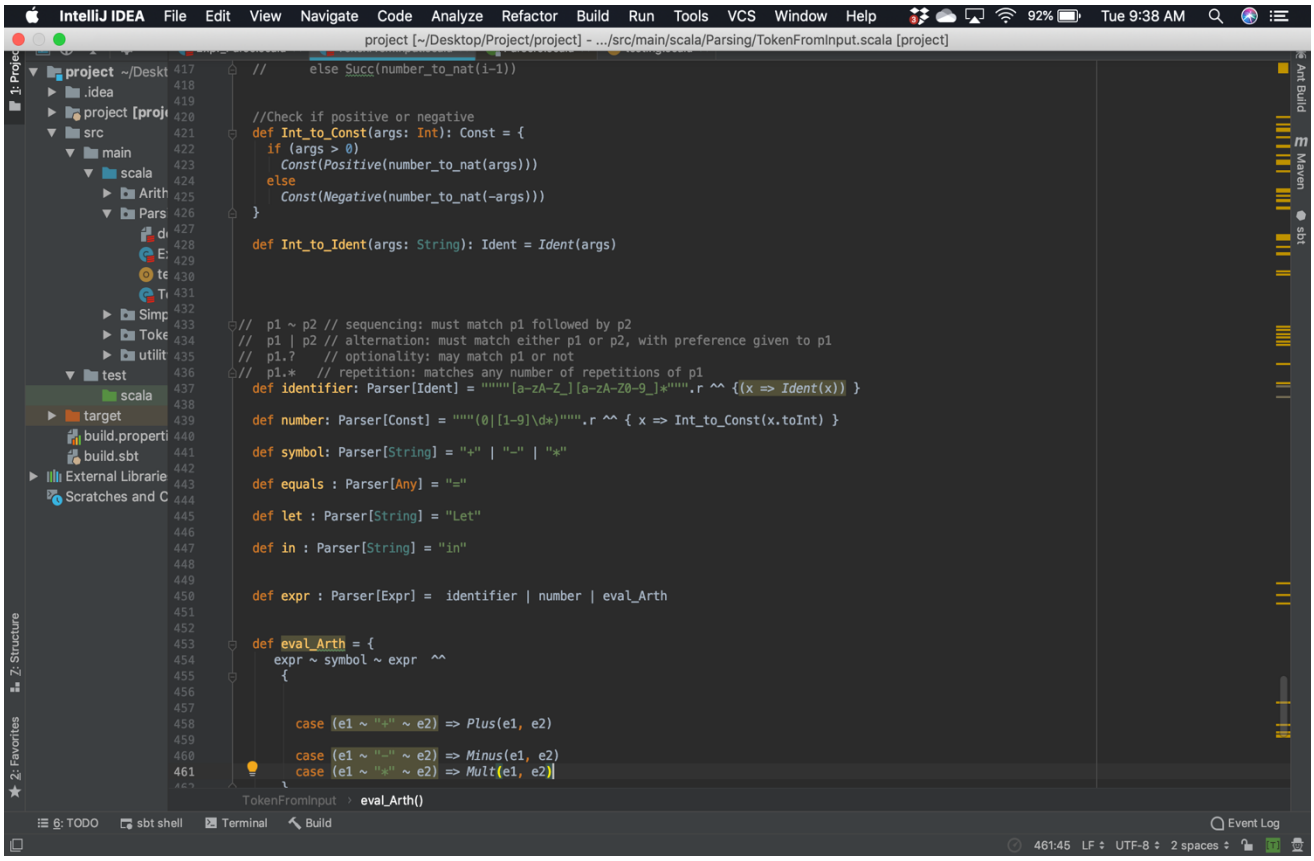
p1 ~ p2 mean sequencing: must match p1 followed by p2 .

p1 | p2 mean alternation: must match either p1 or p2, with preference given to p1.

p1.? mean optionality: may match p1 or not

p1.* mean repetition: matches any number of repetitions of p1

These help me a lot for parsing lettuce. And then I started to write my code, I had struggled with “Import scala.util.parsing.combinator._”. It kept me a error. I don’t understand why, so I had to ask TA (Benno) for help about this before continuing. He could not find out the problem yet. So I decided to download Intelli J IDEA CA for started writing the code. I wrote eval_Arith function that take two expr and symbol (+, -, *, ^), and then I tested (5+2), it’s passed the test.



However, I tested (5+2+3), it's failed. Then I had to do another research for this problem. I could not figure out this problem. I visited Benno's office Hour for help. Benno found the way to make Jupyter Notebook work. So I had to move all my code from Intelli J IDEA CA to Jupyter Notebook. It's much better to use Jupyter Notebook because I can run my code and see output what I want. I asked Benno why my code didn't work for testing several operators (5+2*2) together. He found out that we need to use parentheses for every operator. For example, ((3+2)*4)), (((8-2)+3)-1)

Everything has been working fine for arithmetic. I started to write logic "and" and "or" and "eq" function. Those was so easy to write. However, When I started to do let, FunDef, FunCall and LetRec, it took me more than five hours to get it done. I went to ask Benno for help to write LetRec. Finally, I got all of them work with Check_Assert function that I wrote to test all the function definition.

```
def Check_Assert(x : Expr, expected : Value): Unit = {
        assert(eval(EmptyEnv, x) == expected)
        println("Pass Test!!")
}
```

After that, I researched how to do REPL for extra credit. It's not bad for me. I just create scanner variable, and import "new java.util.Scanner(System.in)" to read user input. I wrote function readInput that take no argument. I use while loop for repeating the user input, and Boolean for stopping the while loop. When I get user input, I parsed input to ParserLettuce class that I wrote and evaluate the output.

```
def readInput {

    var bool_ = true
    var scanner = new java.util.Scanner(System.in)
    println("Welcome to Scala!!!!")
    println("Please enter your input or (quit) to quit")
    while (bool_ == true) {
        var userInput = scanner.nextLine()
        while(userInput =="")
        {
            println("Your input was empty. Please enter your input again.")
            println()
            userInput = scanner.nextLine()
            if(userInput == "quit")
            {
                bool_ = false
            }
        }
        if(userInput == "quit")
        {
            bool_ = false
        }
        else {
            var parsed = ParserLettuce(userInput)
            var eval_ = eval(EmptyEnv, parsed)
            println("Your input is " + userInput)
            println()
            println("Evaluated > " + eval_)
        }
    }
}

```

```
defined function readInput
```

Overall, I have spent more than 10 hours to do this project. I feel I had learned a lot from this project. It's not really difficult as I expected.

Reference:

1. Programming in Scala, Third Edition by Bill Venners, Lex Spoon, Martin Odersky

2. https://github.com/enear/parser-combinators-tutorial/blob/master/src/main/scala/co/enear/parsercombinators/lexer/WorkflowLexer.scala

3. https://www.cs.helsinki.fi/u/wikla/OTS/Sisalto/examples/html/ch31.html

4. https://www.scala-lang.org/api/2.12.2/scala-parser-combinators/scala/util/parsing/combinator/Parsers.html#Input=scala.util.parsing.input.Reader%5BParsers.this.Elem%5D