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)

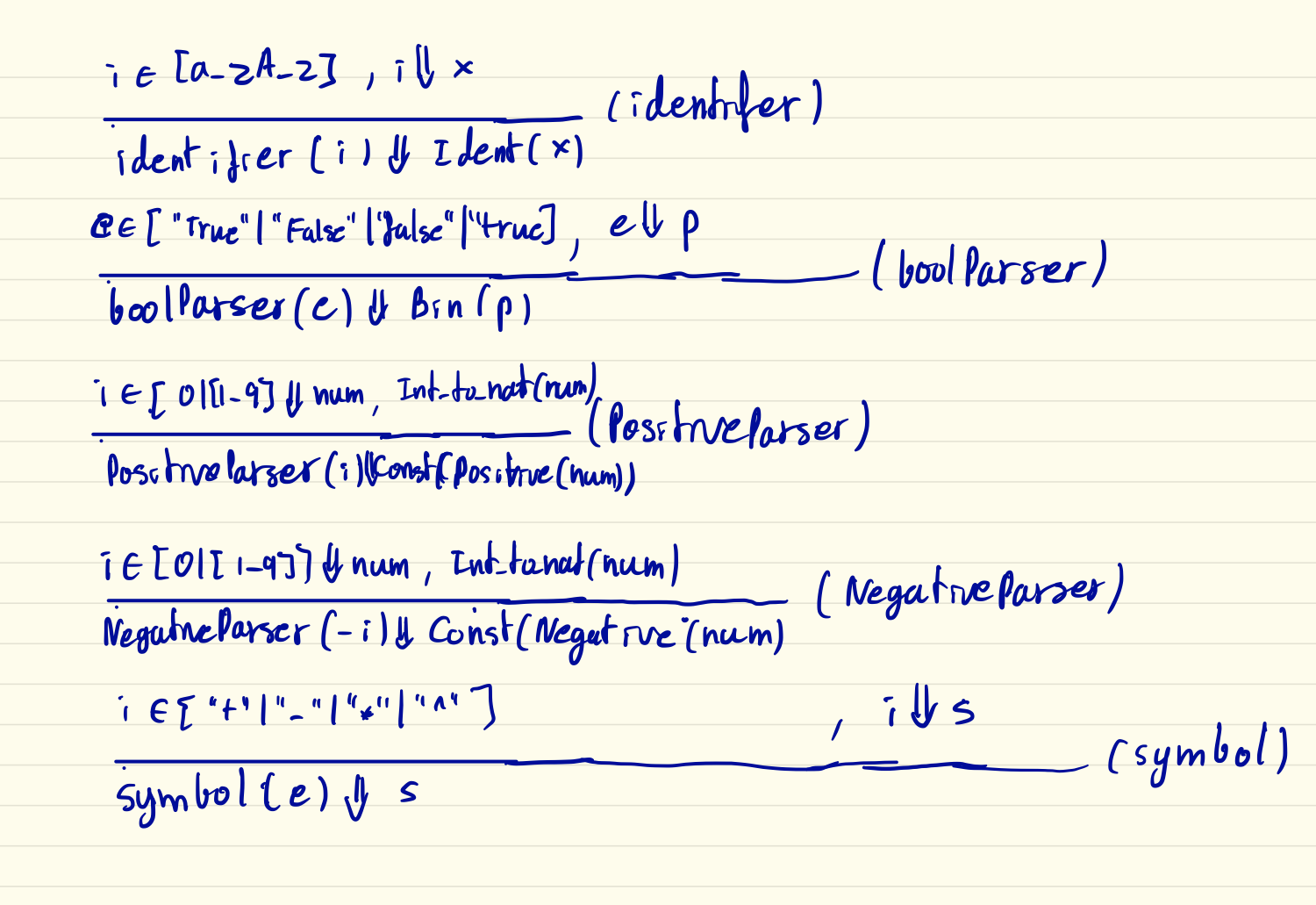
| eval\_Let( Let\_Parser, expr, equals, expr, in\_Parser, expr)

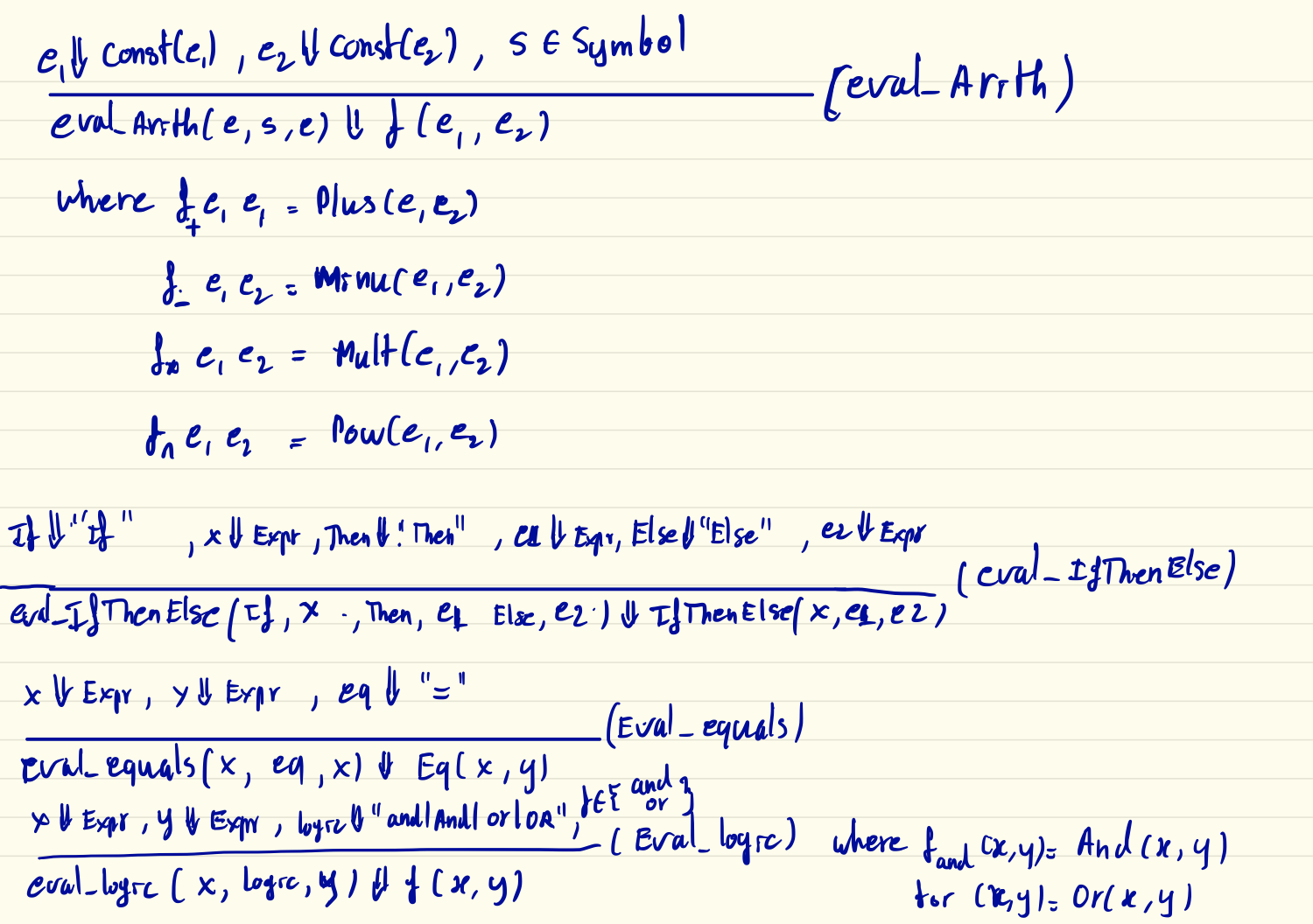
| eval\_FunDef(String, “(“, String, “)”, expr)

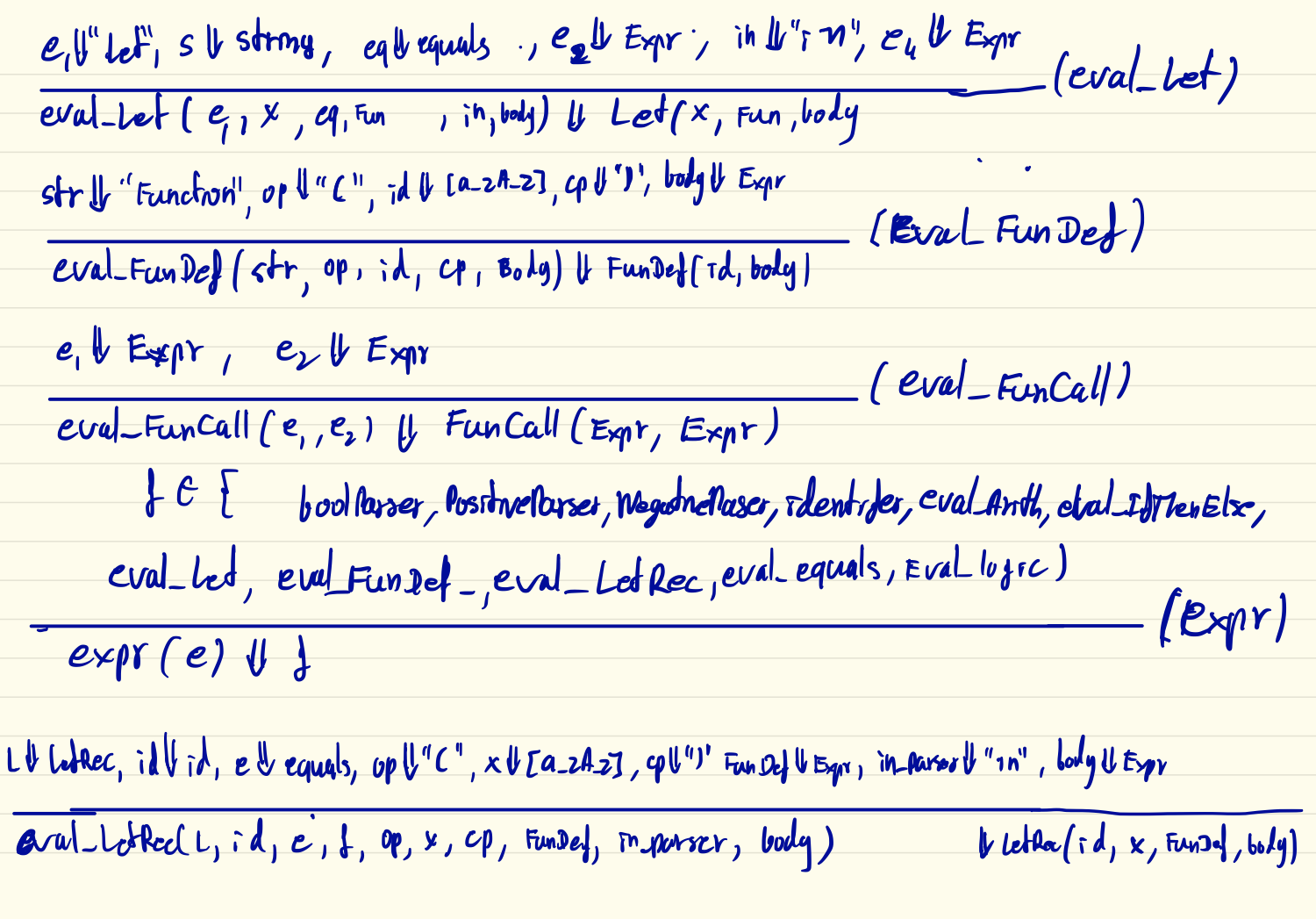
| eval\_FunCall(“(“, expr, “[“, expr ”]”, “)”)

| eval\_LetRec( LetRec, str, equalsl, str, “(“, expr, “)” expr, in\_Parser, expr)

| apply(String)

These are inference rules:





I have learned some symbols like

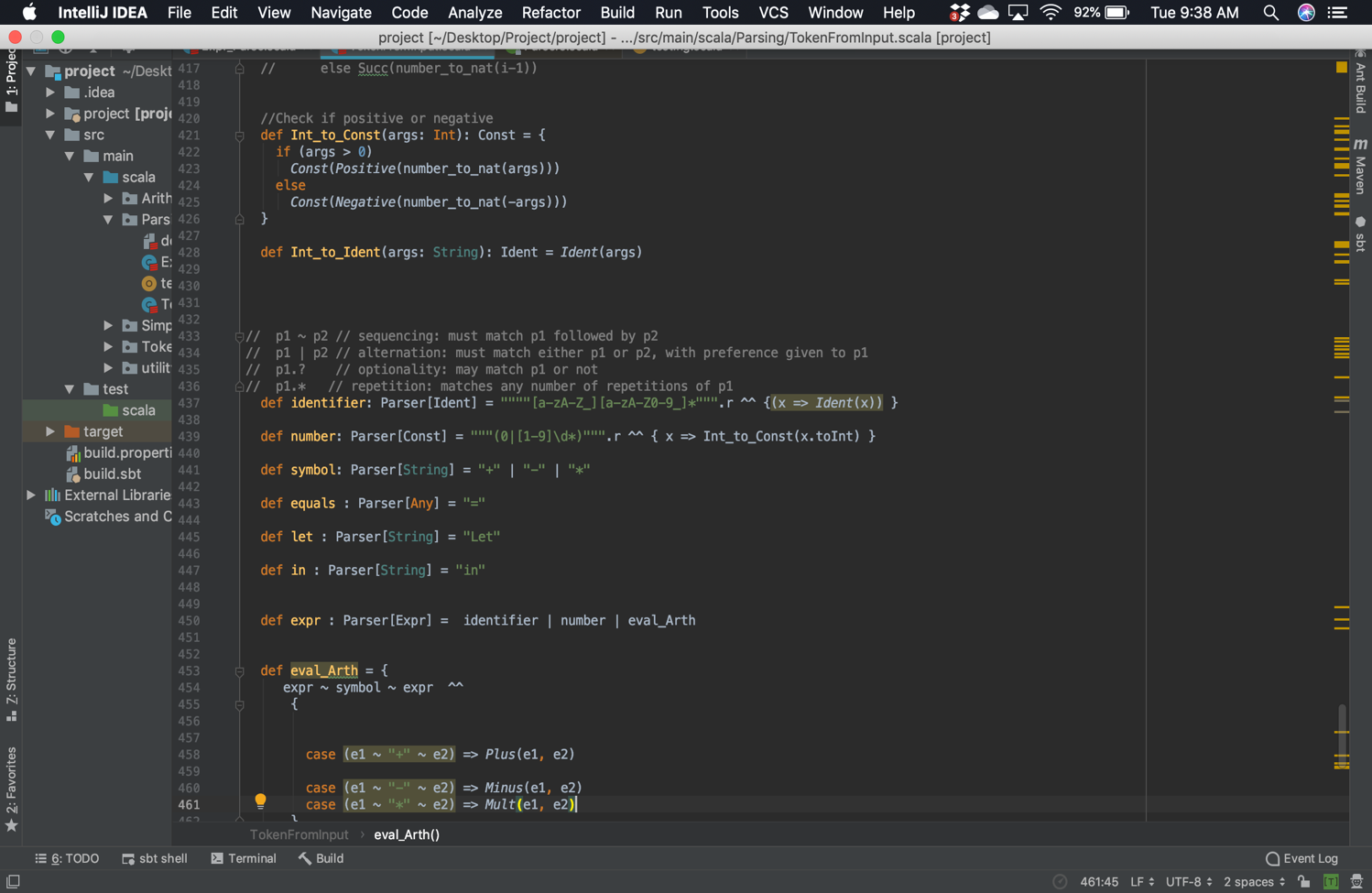
p1 ~ p2 mean sequencing: must match p1 followed by p2 .

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.



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](https://learning.oreilly.com/library/view/programming-in-scala/9780981531687/)
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>